UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

DEPARTAMENTO DE TECNOLOGÍA ELECTRÓNICA

Aplicación en entorno gráfico para el control de un sistema microposicionador motorizado

PROYECTO FIN DE CARRERA

INGENIERÍA TÉCNICA INDUSTRIAL: ELECTRÓNICA INDUSTRIAL

Autor: SEBASTIÁN MORA RUS

Directora: VIRGINIA URRUCHI DEL POZO

A mi tutora Virginia y a mi amigo Agustín.

A mis amigos y compañeros.

A mi madre, hermanos, novia y familia.

Y muy especialmente con todo mi amor y cariño, a mi padre.

Gracias.

ÍNDICE

ÍNCIDE DE FIGURAS

CAPÍTULO 1

INTRODUCCION

1.1 OBJETIVOS GENERALES Y USO DE LA APLICACION

Los sistemas microposicionadores se destinan a ubicar piezas o componentes en posiciones concretas cuya ubicación está prefijada. En particular, esta función resulta de gran utilidad en sistemas de caracterización electroóptica donde la precisión y la repetitividad en el posicionamiento son muy importantes.

En el Grupo de Displays y Aplicaciones Fotónicas del Departamento de Tecnología Electrónica surgió la necesidad de realizar pruebas con sistemas electroópticos para realizar medidas con espaciados micrométricos, lo que derivó en la necesidad de disponer de un sistema microposicionador. En el mercado existen soluciones integradas de microposicionadores que cumplen dicha necesidad, pero todos ellos necesitan del uso de algún *software* de control para hacerlos funcionar a menudo no suficientemente versátiles. Por ello, en un proyecto previo [1], se construyó y desarrolló un sistema microposicionador controlado en un eje para incorporarlo dentro de un sistema óptico de caracterización. El sistema contaba con un bloque mecánico, un bloque *hardware* con los circuitos electrónicos para el control del bloque mecánico y un bloque *software* con el programa de control específico. El sistema al completo puede verse en la Figura 1.1.

La parte mecánica del sistema que fue desarrollado, consta de una vagoneta, un carril por donde se desplaza la misma, un motor paso a paso que moverá la vagoneta a través de un sistema de correa y unos sensores mecánicos de final de carrera colocados en los extremos del carril.

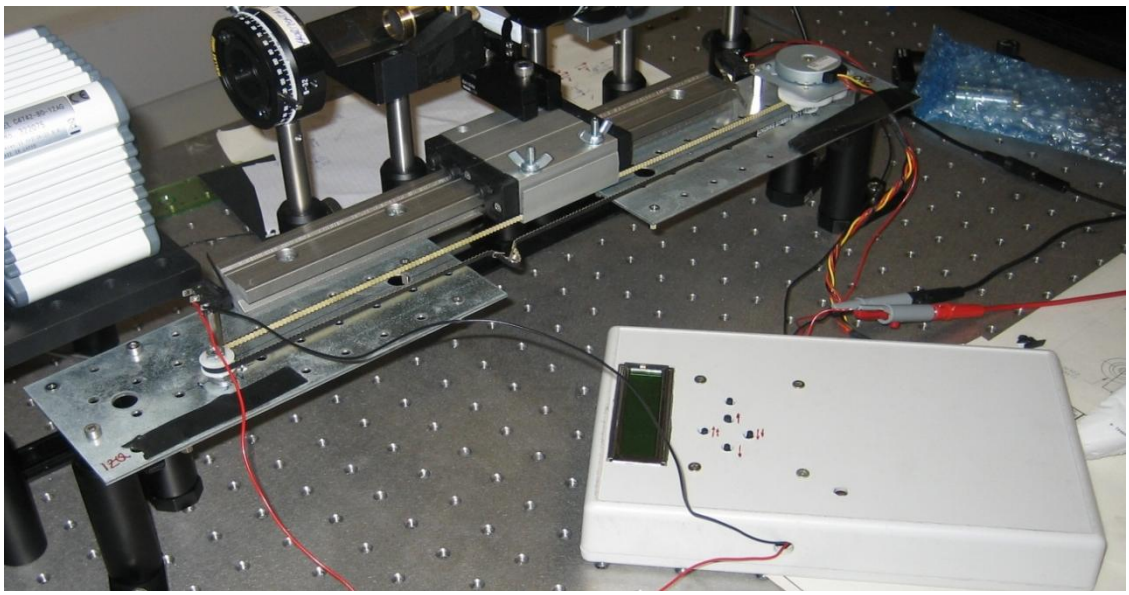


Figura 1.1 Sistema microposicionador desarrollado en un proyecto previo

Al *hardware* del microposicionador se le dotó de un *driver* de control de motores paso a paso, un sistema de pulsadores para hacer mover la vagoneta, una LCD para visualizar en todo momento la posición de la misma y un puerto USB para poder realizar una posterior conexión con un PC, pero no se ahondó en el estudio de la comunicación futura.

Después de haber desarrollado el sistema microposicionador, se decidió profundizar más y abordar la posibilidad de incorporar mejoras en el propio sistema ya creado y la realización de un *software* de control para controlar de forma remota la posición a través de un PC. El proyecto que nos ocupa se formula con ese objetivo general. Además, se desea mejorar algunos aspectos mecánicos, realizar también mejoras en el propio *software* del microcontrolador, y fundamentalmente desarrollar una aplicación en entorno gráfico basada en el programa LabVIEW que controle el sistema microposicionador.

La realización del presente proyecto supone una ventaja añadida respecto al sistema antiguo. El microposicionador previo era capaz de trabajar en modo autónomo, controlando el movimiento de la vagoneta a través de los pulsadores externos. Dicho funcionamiento presentaba la ventaja de la autonomía del sistema sin depender de fuentes externas de control. La principal limitación era que el uso de los pulsadores sólo permitía la implementación de funciones básicas de movimiento y sin posibilidad de definir secuencias. Además el uso de un número limitado de pulsadores hacía poco cómoda la programación del movimiento. En el nuevo *software* que se plantea para este proyecto, el sistema comparte las dos formas de funcionamiento, modo autónomo y modo remoto a través de PC. La gran ventaja en el uso del microcontrolador por control remoto es la comodidad y sencillez en la definición de la posición debido al entorno gráfico de programación utilizado (LabVIEW), muy amigable para el usuario. Además, este entorno de trabajo es muy potente permitiendo realizar bucles de movimiento basados en secuencias predefinidas e incluso implementar la funciones de la pantalla LCD.

A continuación se detallan las partes mecánicas y de *hardware* principales que componían el antiguo microposicionador y, posteriormente, se mostrarán los bloques que se han desarrollado en este proyecto. En primer lugar, el sistema mecánico desarrollado en el proyecto previo se representa a modo de bloques en la Figura 1.2. Se observa que consta de un microcontrolador que se comunica con los dispositivos externos: pantalla LCD, pulsadores, finales de carril y motor. En la Figura 1.3 se muestran las mejoras que se desea añadir a la parte mecánica del sistema previo.

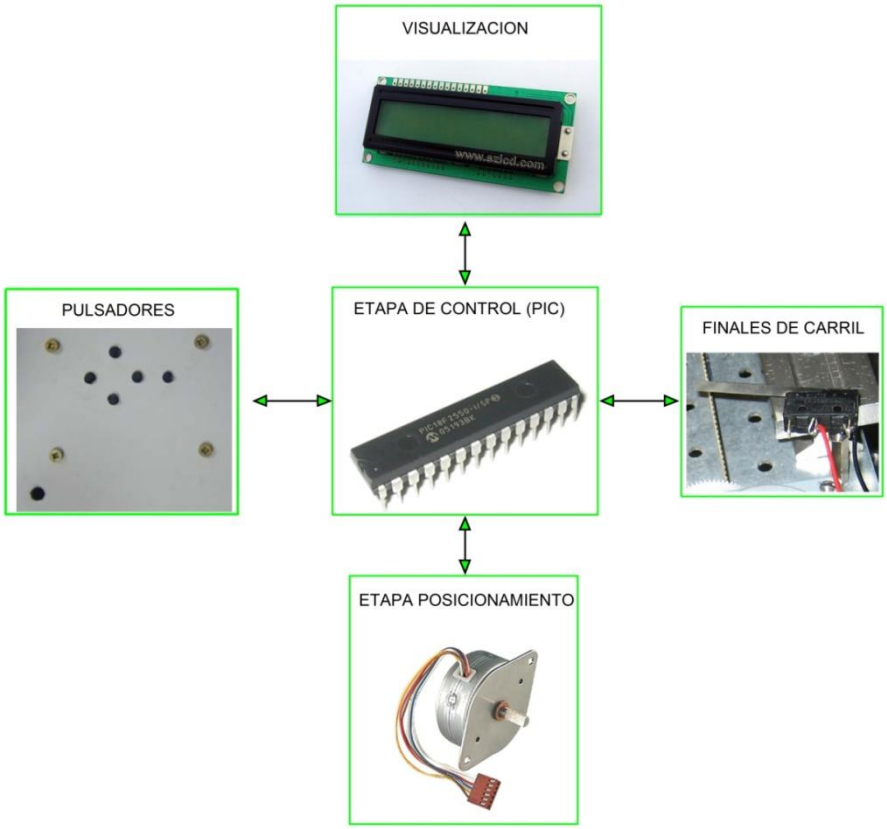


Figura 1.2 Sistema mecánico previo al proyecto actual

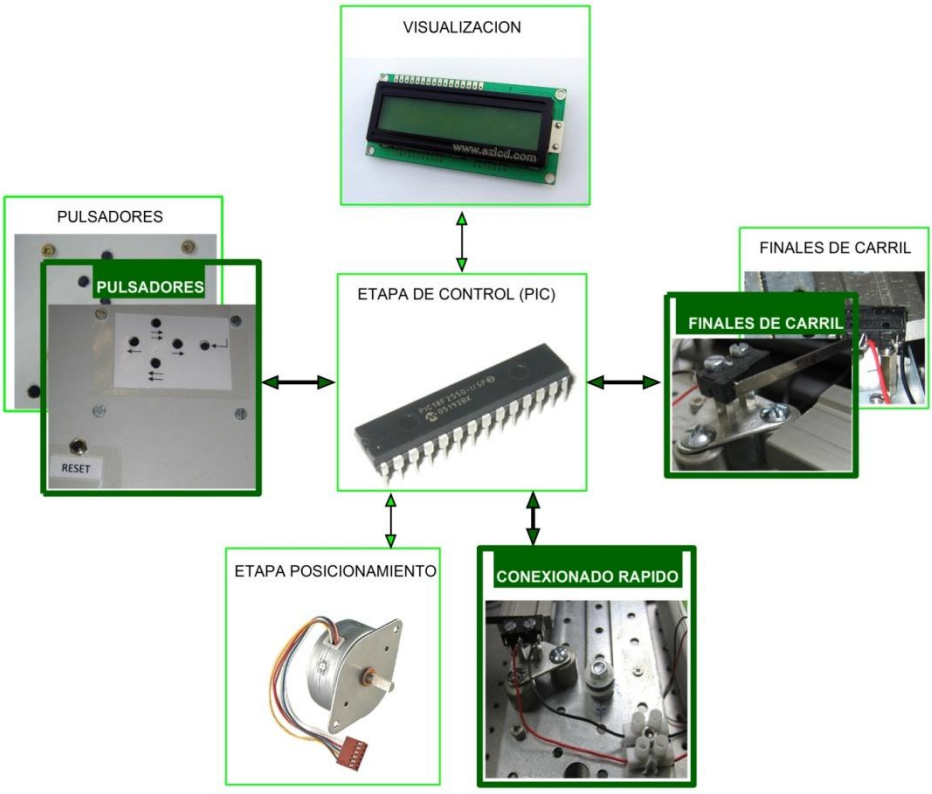


Figura 1.3 Sistema mecánico futuro que será desarrollado

Hay que indicar que se han representado en verde claro los bloques del sistema mecánico previo y se han superpuesto con bloques verde más oscuro los puntos en los que se han incluido mejoras sobre los bloques del diseño previo. En este caso, en el sistema mecánico se muestran, en verde resaltado, varios bloques que se han mejorado con diferente grado de profundidad. En concreto, se han modificado los sistemas de apoyo y sujeción de los finales de carril, se ha simplificado la conexión entre el *hardware* electrónico y la parte mecánica y, finalmente, se ha incluido un botón de RESET accesible desde el exterior y un conjunto de etiquetas identificativas de la función de los pulsadores.

Por otra parte, el programa software implementado en el proyecto previo, que se toma como punto de partida, cuenta con las funciones principales que se muestran en la Figura 1.4. En ella se presentan de forma resumida las 4 funciones principales que desarrollan casi la mayoría del código de control del microposicionador: MAIN, LEER_PULSADORES, ACCIONES_MANUALES Y PASO_MOTOR. En este esquema se han representado los bloques en cajas de color rojo sin resaltar y con flechas las comunicaciones entre bloques.

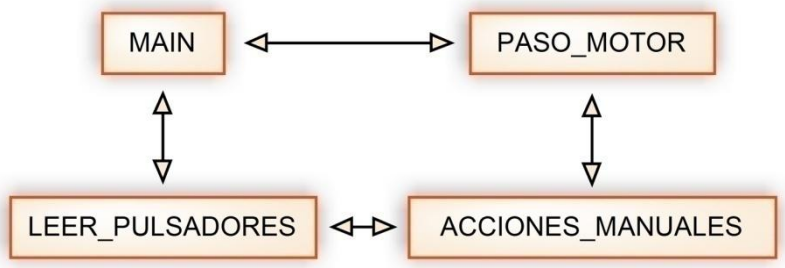


Figura 1.4 Funciones *software* principales en el diseño previo

En la Figura 1.5 se representan en color rojo resaltado los bloques desarrollados en este proyecto, a partir del diseño previo. En este caso, no se muestran cajas superpuestas con las ya programadas sino que, por simplicidad, se han ilustrado todos los bloques separados de los anteriores y comunicados con ellos. La característica principal del programa diseñado, tanto a nivel de microcontrolador, en C, como a nivel de entorno gráfico, en LabVIEW, ha sido su propiedad modular. Es decir, los bloques de código se han agrupado en funciones básicas representativas que serán la base del desarrollo. En particular, la comunicación entre dispositivos vía USB, se abordó mediante programa modular en el que se incluyeron funciones nuevas (como

ESCRIBIR_USB, LEER_USB, ACCIONES_USB u otras) y las antiguas se modificaron mínimamente. Además, la comunicación entre el microcontrolador y el PC, se enfocó a través de la creación de un protocolo de comunicación. Y, finalmente, dicho protocolo serviría de enlace con el programa a diseñar en el entorno gráfico de programación LabVIEW. Hay que destacar que el diseño de la aplicación para PC con el entorno de programación LabVIEW ha supuesto un reto añadido, que ha incluido una etapa inicial de aprendizaje en la programación del nuevo lenguaje llamado “Lenguaje G”.

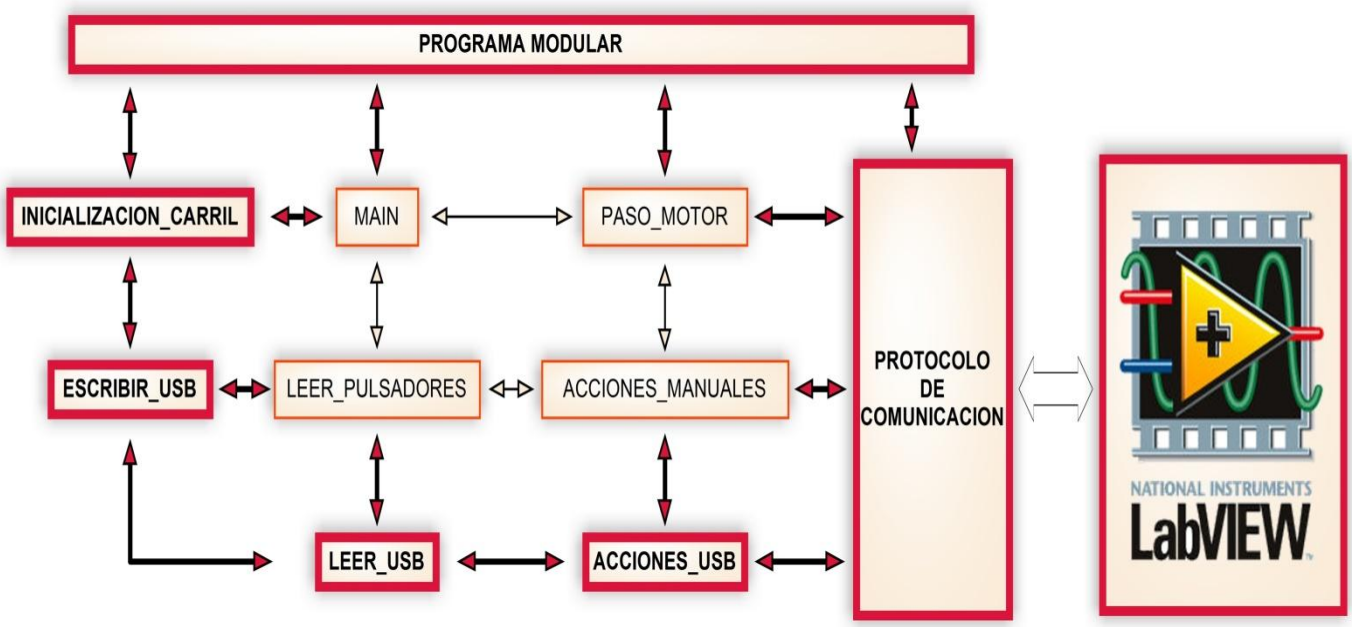


Figura 1.5 Funciones nuevas protocolo y *software* de control en entorno gráfico

Por tanto, el objetivo general del presente proyecto es mejorar algún aspecto mecánico y, fundamentalmente, *software* del microposicionador, y ante todo, dar una solución alternativa al modo autónomo del microcontrolador diseñando una interfaz capaz de comunicarse, controlar y convivir con todo el sistema mediante el programa LabVIEW. Los objetivos específicos del proyecto según todo lo comentado anteriormente, se incluyen en el Capítulo 2.

1.2 ESTRUCTURA DEL DOCUMENTO

A lo largo de este documento de memoria del proyecto desarrollado se describirá el diseño, desarrollo e implementación del *software* necesario para comunicar el sistema

microposicionador con un PC. A continuación se listan los contenidos de cada uno de los capítulos de forma resumida.

- En el Capítulo 2 se expone la motivación y los objetivos específicos para el desarrollo del Proyecto.
- En el Capítulo 3 se enumeran y explican las funciones desarrolladas en el entorno gráfico de programación, LabVIEW, y el protocolo de comunicación escogido para interconectar ambos sistemas.
- En el Capítulo 4 se expone la programación en C del microcontrolador que se ha desarrollado para hacerlo más funcional, y la comunicación con el PC. Mediante sencillos diagramas de flujo se describen las tareas que realizan las funciones implementadas en el programa.
- En el Capítulo 5 se muestra una guía de utilización y verificación del programa desarrollado, y se enumeran las mejoras introducidas al sistema mecánico previo.
- En el Capítulo 6 se describen las conclusiones obtenidas en el desarrollo de este Proyecto y las ampliaciones futuras.

Asimismo, se incluyen al final del documento los apartados normalizados: Listado de la bibliografía consultada, Anexos, Planos, Pliego de condiciones y Presupuesto.

CAPÍTULO 2

MOTIVACIÓN Y OBJETIVOS ESPECIFICOS

2.1 MOTIVACIÓN

La principal motivación para el diseño del *software* de control, es la necesidad de poder controlar el sistema previo mediante PC y aprovechar las ventajas que esto supone, con el entorno gráfico de programación LabVIEW. Se simplificará el uso del microposicionador por el usuario final gracias a la interfaz amigable. Y se dejará abierto el camino de la programación de secuencias complejas de movimiento no conseguidas con las herramientas previas de programación.

El establecimiento de mejoras a partir del proyecto previo desarrollado programado en C, supone otra motivación importante propia del desarrollo en sí del trabajo. Este planteamiento de continuidad y mejora ha supuesto un estímulo para ahondar en las características de programación del microcontrolador PIC18F2550 para el desarrollo de la nueva implementación.

Finalmente, en la línea de aprendizaje y formación derivada del trabajo a realizar, el uso del entorno gráfico de programación LabVIEW ha supuesto una motivación muy interesante ya que esta herramienta *software* de programación se utiliza en grandes empresas en la industria y su uso es compartido por ingenieros altamente cualificados.

2.2 OBJETIVOS ESPECÍFICOS

Una vez que se han establecido los objetivos generales de este proyecto en el Capítulo 1, se describen a continuación los objetivos específicos que guiarán el desarrollo del mismo. Se han planteado estos objetivos en una lista que pretende resumir los aspectos mejorados en la parte mecánica del sistema y los bloques rediseñados y nuevos tanto en el *software* del microcontrolador, como en la interfaz grafica de usuario final.

MEJORAS DEL SISTEMA MECÁNICO Y HARDWARE ELECTRÓNICO

- ✓ Diseñar un sistema de anclaje alternativo al ya existente para los finales de carrera.
- ✓ Sustituir el botón de RESET, no accesible desde el exterior, por uno de superficie.

- ✓ Incluir pegatinas informativas que muestre la función que desempeña cada pulsador.

- ✓ Dotar a los finales de carril y al motor de un conexionado rápido y sencillo.

OBJETIVOS ESPECÍFICOS DE LA PROGRAMACIÓN EN C

A partir del programa en C ya desarrollado se pretende:

- ✓ Reorganización del *software* del microcontrolador para dotarlo de funciones modulares que ayuden al programador para ampliaciones futuras.

- ✓ Realizar una inicialización del carril (con un barrido de la longitud del mismo), guardando la posición de ambos extremos para su futura utilización y posibilidad de calibrado de nuevos carriles. Permitir el uso versátil del carril, acotando su longitud total de trabajo según la aplicación del usuario final.

- ✓ Mejorar el sistema de visualización LCD evitando el parpadeo del mismo.

- ✓ Aumentar la versatilidad del control del motor permitiendo la programación de la velocidad de búsqueda para movimientos largos de la vagoneta.

- ✓ Realizar un protocolo de comunicación para comunicar el microcontrolador y el PC.

- ✓ Dotar al sistema de medidas de protección para evitar daños en los componentes circundantes a la vagoneta. Se propone la parada inmediata del movimiento del motor mediante el sensado de la pulsación de los finales de carrera acoplados en los extremos del carril, sin repercutir de manera negativa en ningún parámetro.

OBJETIVOS ESPECÍFICOS DE LA PROGRAMACIÓN EN LABVIEW

Realizar un *software* de control para el microposicionador que incluya los siguientes puntos:

- ✓ Diseñar un protocolo de comunicación bidireccional que sea capaz de comunicar el PC con el microcontrolador.

- ✓ Dotar a la aplicación de un botón de inicialización de carril para no tener que reiniciar el microposicionador cada vez que se quiera ejecutar esta acción.

- ✓ Simplificar la definición de la posición de destino de la vagoneta, liberando al usuario de la realización de ningún cálculo y proteger al sistema de posibles errores en la definición del movimiento.

- ✓ Dotar a la aplicación de un sistema sencillo y automático de posicionamiento de la vagoneta, introduciendo simplemente 2 parámetros de control (Número constante de posiciones a mover y posición a la que se quiere enviar la vagoneta). Poder definir la velocidad máxima de movimiento de la vagoneta.

- ✓ Ofrecer al usuario la posibilidad de definir de forma sencilla secuencias repetitivas de movimiento del motor a modo de bucles y de establecer los tiempos de espera entre bucles.

- ✓ Conservar el modo doble de establecimiento de la posición: modo autónomo a través de pulsadores y modo remoto a través de PC.

- ✓ Permitir la monitorización, en tiempo real, de la posición de la vagoneta y de los finales de carril, e incluir visualizadores para revisar los datos enviados y recibidos por el puerto USB.

CAPÍTULO 3

DISEÑO DE LA APLICACIÓN EN ENTORNO GRAFICO

3.1 INTRODUCCION AL ENTORNO GRAFICO DE PROGRAMACION LABVIEW

En este capítulo se profundiza en el diseño del *software* de control que permitirá la comunicación del programa de entorno grafico con el microcontrolador. Para realizar las tareas de programación especificas, se justificará el entorno gráfico de programación elegido, se enumerarán cada una de las partes principales del código desarrollado, y se ahondará en la descripción de cada una de ellas. A continuación se resume brevemente cada uno de estos aspectos.

1. Un *entorno grafico* de programación, es un programa que se estructura a través de bloques visuales que interactúan entre ellos facilitando la labor del programador.

2. Al conjunto de interacciones lógicas basadas en instrucciones que ha de ejecutar un ordenador para realizar tareas específicas, se le denomina *programa*.

3. Al proceso por el cual se diseña, escribe, prueba, compila y depuran posibles errores, se le llama *programación*.

3.1.1 ELECCIÓN DEL PROGRAMADOR

En la actualidad existen varios programas que se ajustan a las necesidades del proyecto, es decir, crear una interfaz grafica que se comunique con el sistema microposicionador. Algunos ejemplos de ellos son: Visual Basic, LabWindows, Matlab, LabVIEW, C++, etc.

A continuación se describen brevemente algunos de los aspectos por los cuales se ha escogido el entorno grafico en LabVIEW (*Laboratory Virtual Instrument Engineering Workbench de National Instruments*) para diseñar la aplicación:

1. Es sencillo de manejar, muy ilustrativo y flexible para futuros cambios. Está basado en un nuevo sistema de programación gráfica llamada lenguaje G.

2. Es un programa enfocado a la instrumentación virtual, por lo que cuenta con numerosas herramientas de presentación: gráficas, botones, indicadores,

controles, etc. ahorrando con ello mucho tiempo de programación. La realización de dichas funciones en entornos de programación como C++, sería más complicada e implicaría una inversión de tiempo mayor para lograr el mismo objetivo.

3. Permite una fácil integración con componentes *hardware* como: tarjetas de medición, adquisición y procesamiento de datos, ya sean adquiridas a *National Instruments*, propietaria del entrono gráfico de programación LabVIEW, o a otro fabricante.

4. Es muy completo ya que cuenta con librerías especializadas, como por ejemplo, para manejo de DAQ, redes, comunicación con bases de datos, comunicación con dispositivos externos, control de instrumentos, análisis de datos, creación y modificación de archivos, etc.

5. Es compatible con herramientas de desarrollo similares y puede trabajar simultáneamente con programas específicos de otras áreas de aplicación, como Matlab o Excel. Además, se puede ejecutar en el entorno de varios sistemas operativos diferentes, incluyendo Windows y UNIX, siendo el código transportable de uno a otro.

3.2 CONFIGURACION DE LA PROGRAMACION

La forma de programar en LabVIEW es muy distinta a otros lenguajes de programación que se basan en texto, como Visual Basic, C, etc. La principal diferencia con respecto a los anteriores, es que LabVIEW utiliza símbolos gráficos denominados iconos para representar las acciones.

Los programas de LabVIEW se denominan VI ó VI's (Virtual Instruments), instrumentos virtuales (programas), dado que su aspecto y sus acciones reproducen instrumentos tradicionales como multímetros, osciloscopios, botones, generadores de señales, etc. [2]

A continuación, se explican con detalle los pasos a seguir para la creación de un nuevo proyecto.

Una vez abierto el programa ("Inicio → Programas → National Instruments → LabVIEW 2010 → LabVIEW"), aparecerá una ventana como la mostrada en la Figura 3.1. Entonces se procederá a crear un nuevo proyecto pinchando en "Empty Proyect".

Figura 3.1 Ventana de LabVIEW para seleccionar un nuevo proyecto

Una vez abierto el nuevo proyecto se creará un VI, siguiendo la secuencia "File → New VI", tal como muestra la Figura 3.2.

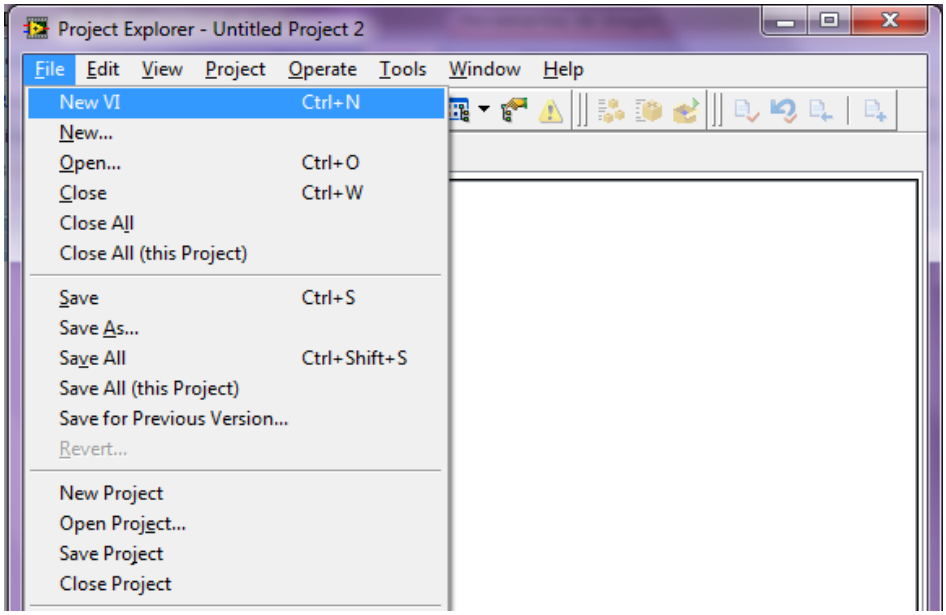


Figura 3.2 Creación de un nuevo instrumento virtual VI

El nuevo instrumento virtual tendrá un aspecto como el de la Figura 3.3, donde se aprecia que está compuesto por dos ventanas: la primera se llama “Front Panel” y la segunda “Block Diagram”.

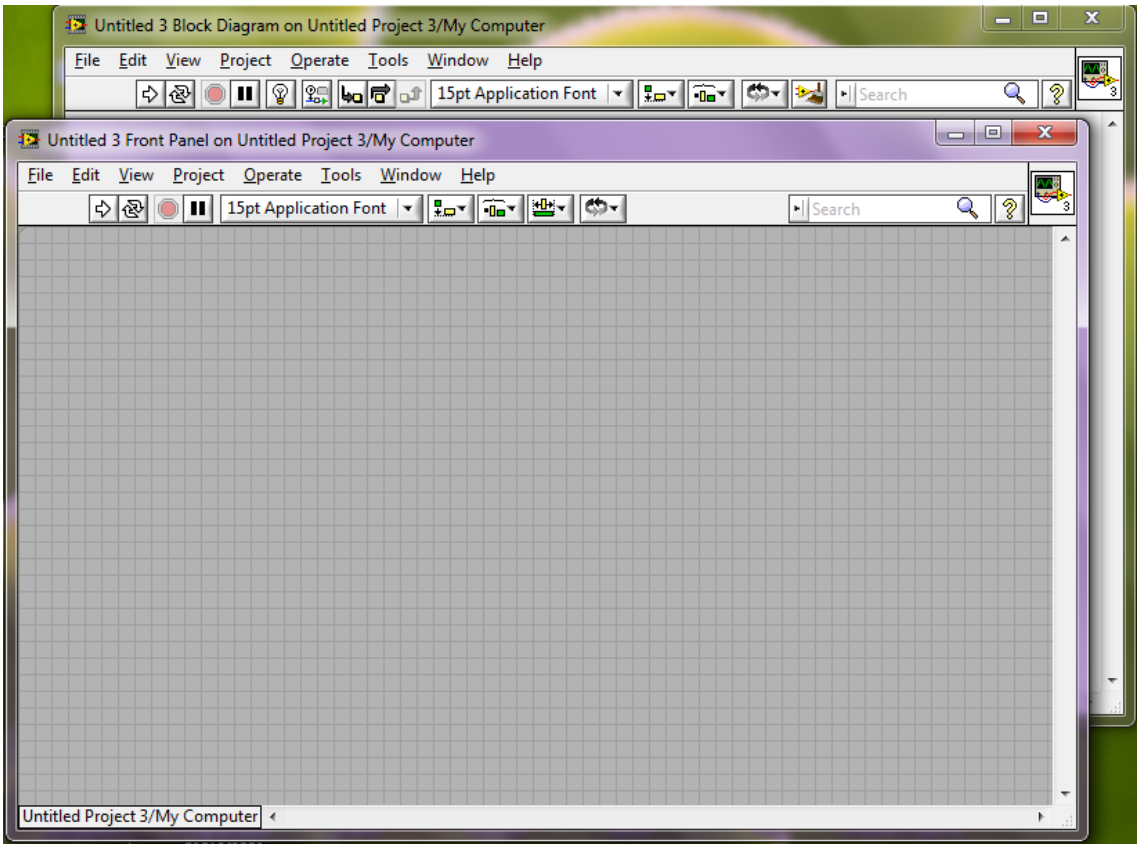


Figura 3.3 Ventana típica de un instrumento virtual VI

En el panel frontal, se diseña la interfaz con el usuario, que contendrá los elementos que van a caracterizar el programa. En él se visualizan, controlan y manipulan los datos. En cambio, en la ventana del diagrama de bloques se aprecia la estructura interna del programa, su algoritmo lógico, en el cual los datos “fluyen” de un bloque a otro a través de las líneas de conexión.

En la Figuras 3.4 y 3.5, se muestra a modo de ejemplo un programa ya creado en LabVIEW incluyendo la ventana del panel frontal y del diagrama de bloques. [3]

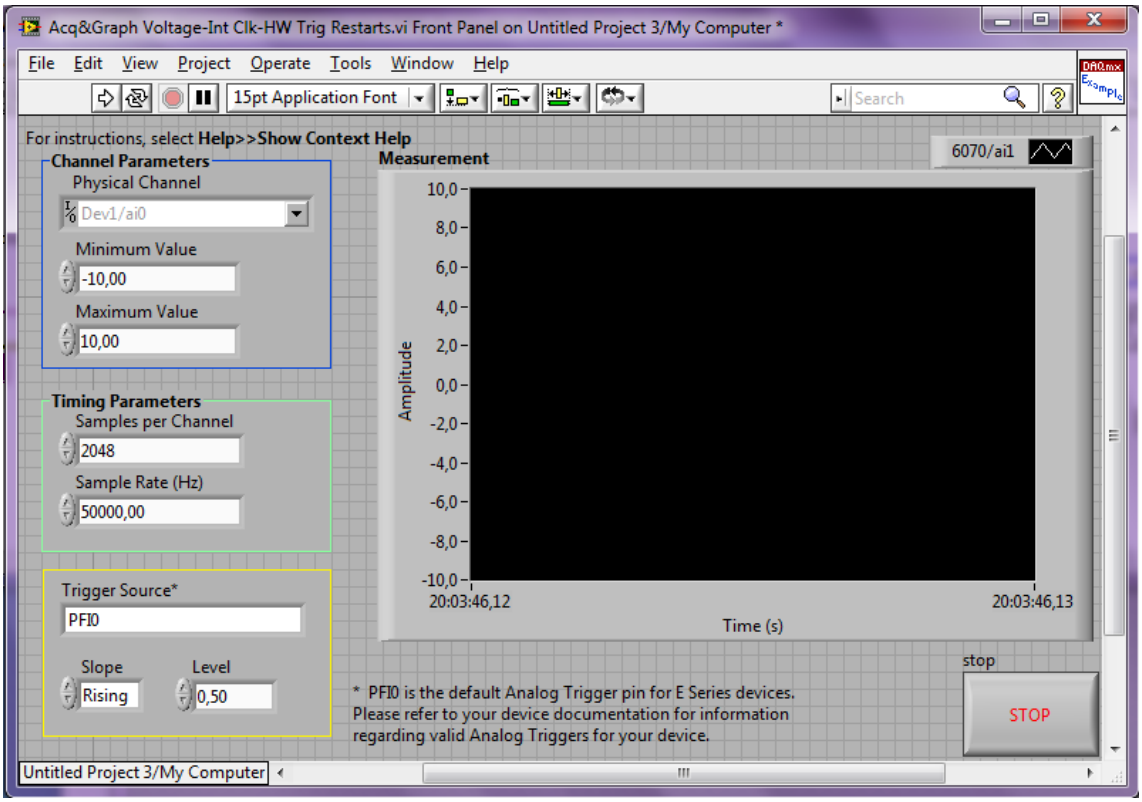


Figura 3.4 Ejemplo de Panel Frontal de un instrumento virtual

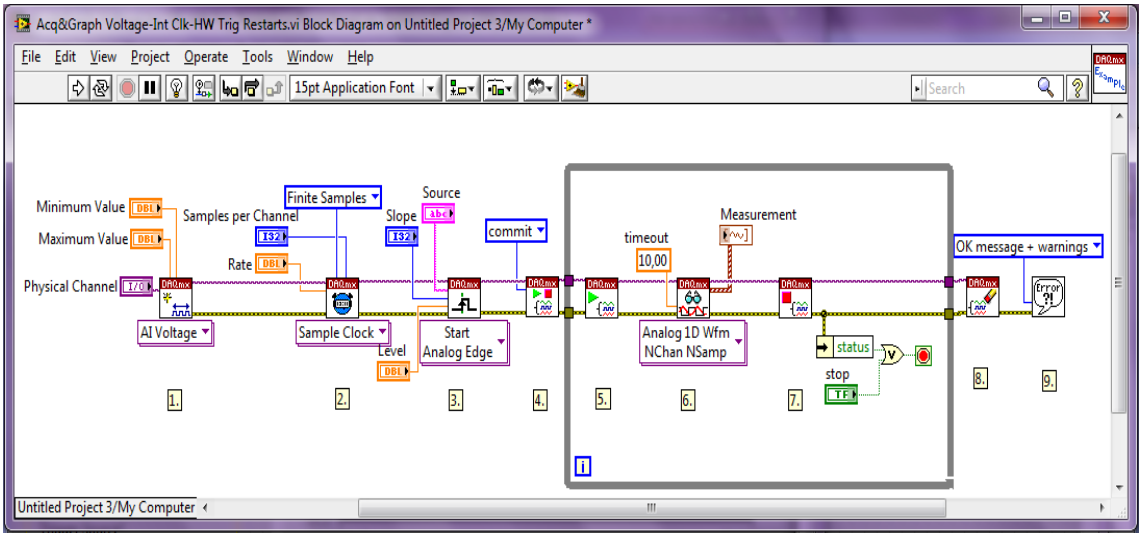


Figura 3.5 Ejemplo de Diagrama de Bloques de un instrumento virtual

Una vez generado el VI, y con él la aplicación programada, se debe guardar en el directorio y con el nombre que se desee, seleccionando para ello “File → Save As…” y eligiendo la dirección de destino. Cuando se haya guardado el VI, se deberán ejecutar los mismos pasos para guardar el proyecto.

Una vez creado el programa, el propio LabVIEW se encarga de compilarlo, dando un mensaje de error si detecta algún error en el código. En la Figura 3.6 se aprecia un ejemplo de mensaje de error.

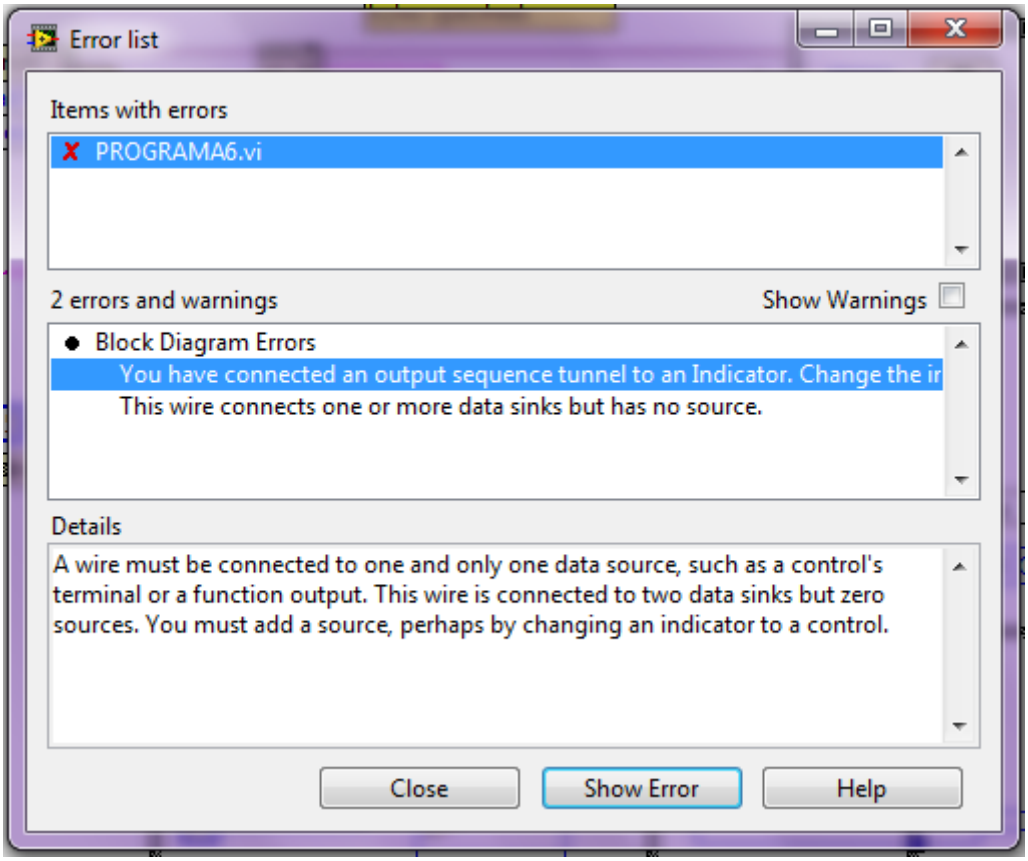


Figura 3.6 Mensaje de error en el código del LabVIEW.

Cuando el LabVIEW no muestra más errores, se puede ejecutar y probar la funcionalidad del programa a través de la pestaña “Operate → Run” o simplemente cliqueando encima del icono en forma de flecha que se muestra en la Figura 3.7.

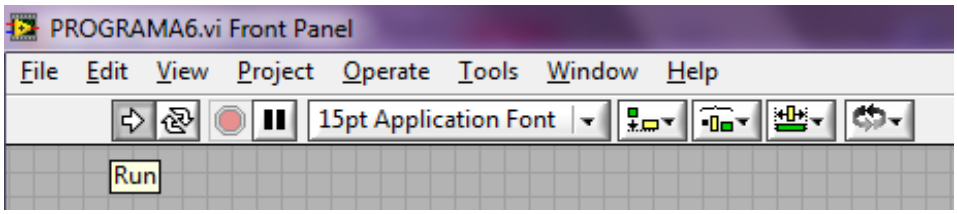


Figura 3.7 Ejecución de un programa en LabVIEW

Si se quiere que el programa se ejecute indefinidamente, se debe pulsar la flecha doble, tal como se muestra en la Figura 3.8.

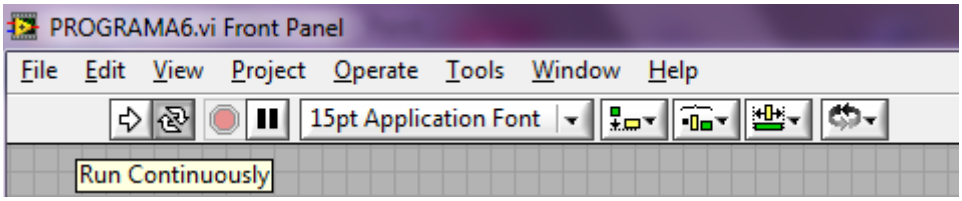


Figura 3.8 Ejecución continua de un programa en LabVIEW

Por último, si el programa no incluye botones para parar la aplicación desarrollada, existe un botón que abortar la ejecución, como se observa en la Figura 3.9. Sin embargo, detener la ejecución mediante el botón de abortar no es una práctica recomendable, ya que se interrumpe el programa de forma brusca y puede acarrear posibles fallos en ejecuciones futuras.

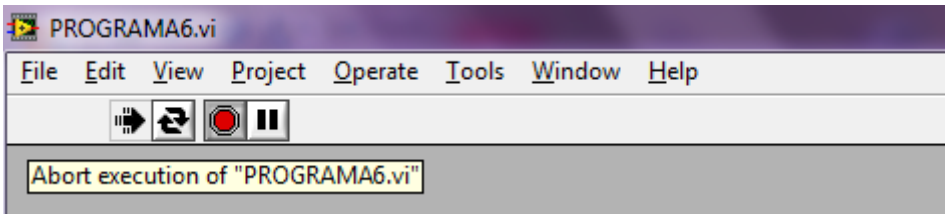


Figura 3.9 Abortar un programa en LabVIEW

3.3 APLICACIÓN DESARROLLADA EN ENTORNO GRÁFICO LABVIEW

En este apartado, se propone y explica el diseño y la programación de la aplicación desarrollada en LabVIEW. Se describen los bloques del programa principal y las funciones esenciales que realiza cada uno de ellos. Para llevar a cabo el proyecto, se han tenido en cuenta dos especificaciones generales de diseño:

- La primera, diseñar una aplicación para comunicar el PC con el microcontrolador mediante un protocolo sencillo que permita del intercambio de información.[4]
- La segunda, implementar bucles repetitivos de las secuencias de movimiento del motor, dotando a la aplicación de la funcionalidad para definir esperas entre ejecuciones del bucle. El objetivo de habilitar un tiempo de espera configurable entre bucles es permitir al usuario realizar otros procesos en secuencia con el

movimiento, como pueden ser medidas en el sistema electroóptico o capturas de imágenes del mismo.

3.3.1 DIAGRAMA DE BLOQUES DE LA APLICACIÓN

El programa principal de la aplicación se puede estructurar en un conjunto de bloques de programa que se comunican entre sí. Durante la ejecución de la aplicación, el PC se comunica con el microcontrolador de forma repetida mediante LabVIEW. Se ha considerado de especial interés para el diseño de la aplicación, la identificación en el programa de los puntos en los que se produce intercambio de información por la comunicación entre dispositivos. Para distinguir estos instantes de forma sencilla, se insertará en la estructura de bloques general el símbolo mostrado en la Figura 3.10.

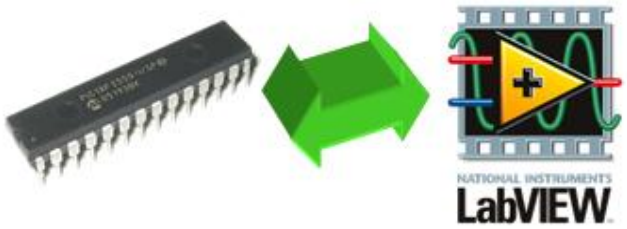


Figura 3.10 Símbolo para identificar de la comunicación entre el microcontrolador y el PC mediante la aplicación de LabVIEW

En la Figura 3.11 se representa el diagrama de bloques a más alto nivel del programa desarrollado en LabVIEW. El programa está compuesto por 5 bloques.

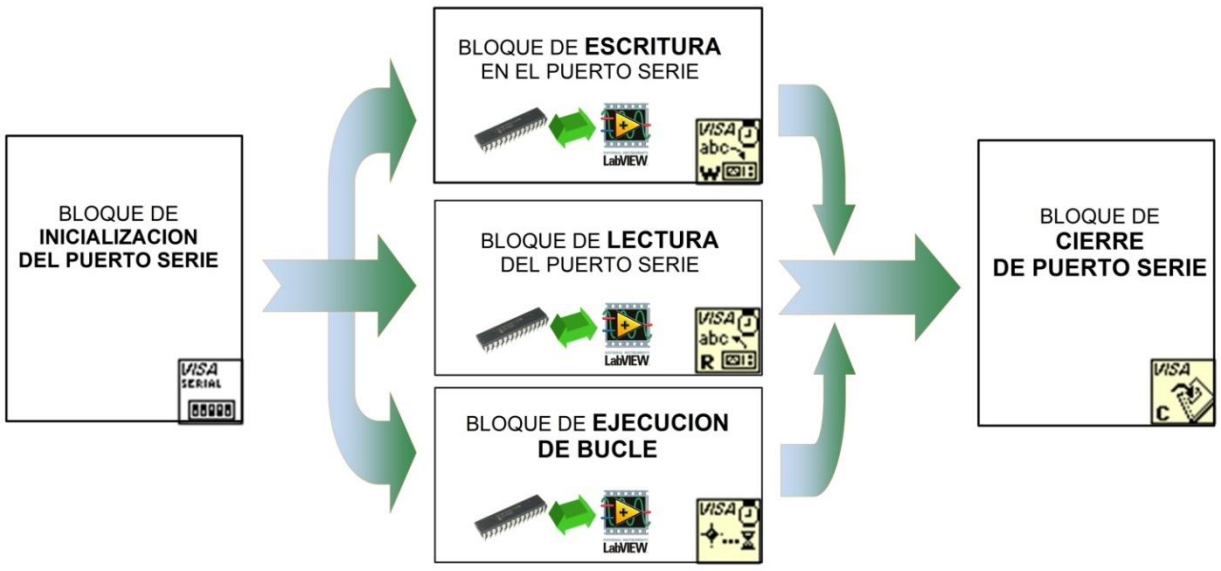


Figura 3.11 Esquema general del programa desarrollado en LabVIEW

El flujo de ejecución comienza por la inicialización del puerto serie (bloque de la izquierda en la Figura 3.11). A continuación, se ejecutan los tres bloques centrales del programa de manera indefinida, hasta que el usuario final decida parar la aplicación. En este caso, la aplicación ejecutará el último bloque y cerrará el puerto serie (bloque de la derecha en la Figura 3.11). Cabe destacar que cualquier programa implementado en LabVIEW se ejecuta de manera no secuencial, es decir, se desconoce qué bloque se ejecutará primero si no se programa explícitamente. Por ello, el flujo es secuencial en todo el proyecto excepto en los tres bloques centrales, cuya ejecución se produce en paralelo, de ahí la distribución de los bloques en la representación mostrada. Se eligió este tipo de programación porque resulta mucho más eficiente que una en modo secuencial.

El primer bloque y el último están destinados a la apertura y el cierre correcto del puerto serie. La ejecución de estos dos bloques de programa hace posible la comunicación entre el PC y los dispositivos externos. Los tres bloques restantes están agrupados en lo que se denomina programa principal, “Main”, a semejanza de la función principal en el código de un programa escrito en C. En esta parte del código se programan las tres funciones siguientes:

- Escritura desde el PC en el puerto serie para ser leído por el microcontrolador.
- Lectura desde el PC de lo que el microcontrolador envíe al puerto serie.
- Ejecución del bucle repetitivo descrito anteriormente.

En capítulos posteriores se profundizará con mayor detalle en cada uno de los bloques generales de los que consta el programa.

3.3.2 PROTOCOLO DE COMUNICACIÓN ENTRE DISPOSITIVOS

Para establecer la comunicación entre los dispositivos del sistema de forma normalizada, se ha considerado conveniente crear y definir un protocolo específico de comunicación por el puerto USB para la aplicación que nos ocupa. Dado que los instrumentos que se conectarán mediante USB no van a convivir en un entorno hostil (industria, intemperie, etc.), se decidió implementar un protocolo lo más sencillo posible.

El protocolo se basa en la definición de una trama de bits que llevan la información para establecer la comunicación. Dicha trama de bits se ha estructurado en

subconjuntos de 8 bits (*bytes*). La longitud óptima de la trama se ha optimizado a 5 *bytes*. Este número de *bytes* era necesario y, al mismo tiempo, suficiente para intercambiar toda la información entre los dispositivos, asegurando con ello su correcto funcionamiento. A continuación se detalla porqué se eligieron 5 *bytes* y no un número menor o mayor. Detallar que los números en binario, llevarán un identificador asociado previo que será el “0b”, y los hexadecimales uno que será el “0x”.

1. Son necesarios 2 *bytes* que indiquen el comienzo y el final de cada trama de datos. En lugar de elegir un *byte* al azar tanto para iniciar la trama como para finalizarla, se decidió utilizar para cada caso el estándar en ASCII.
   - ✓ Para el inicio de trama: STX (*Start of text*) correspondiente al número 1 en decimal y al 0b00000001 en binario.
   - ✓ Para el fin de trama: ETX (*End of text*) correspondiente al 3 en decimal y 0b00000011 en binario.

2. También es preciso enviar algún tipo de información de la longitud del carril en número de pasos o de la posición en tiempo real de la vagoneta sobre el carril. En un proyecto previo [1], se realizaron mediciones sobre la longitud del carril. Como resultado de dichas medidas se obtuvo que el número de pasos del carril estaba en torno a 5200 pasos. Un *byte* no es suficiente para codificar cualquiera de las posibles posiciones de la vagoneta sobre el carril, ya que con un *byte* sólo hay posibilidad de codificar 256 posiciones. Por ello, se eligieron 2 *bytes* para enviar esta información. Con 2 *bytes* (65535 codificaciones posibles) se cubren los 5200 pasos de la longitud del carril. Además, sobran 60335 pasos por si en un futuro se decidiese ampliar el carril, no llevando consigo un cambio en el protocolo de comunicación.

3. Por último, se necesita enviar órdenes para que cada dispositivo sepa que acción ejecutar. Para ello se escogió 1 *byte*, que indica al receptor del mismo qué instrucción debe ejecutar. Las órdenes que se han codificado con ese *byte* dependen del receptor que la vaya a procesar. Si el receptor es el microcontrolador, hará una distinción entre: Velocidad de desplazamiento, inicialización del carril y sentido del movimiento; si el receptor es el LabVIEW, solamente distinguirá entre posición actual de la vagoneta sobre el carril y la posición del extremo derecho.

El aspecto visual de la trama viene representado en la Figura 3.12. Se puede apreciar que cada trama comienza con STX y acaba con ETX. Seguidamente a STX se envía el código de acción, que será el que lleve las acciones a ejecutar. Posteriormente se envía el *byte* de DATA HIGH y el de DATA LOW, que son los encargados de llevar la información de la posición de la vagoneta, o del final del carril, dependiendo de cuando se envíe la trama, y por último se concluye con ETX, que llevará asociado el fin de trama.

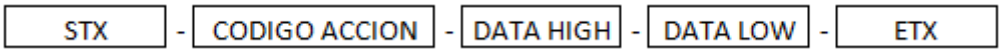


Figura 3.12 Trama de comunicación entre dispositivos

Seguidamente se detalla el equivalente de cada uno de los *bytes* de la comunicación entre los dispositivos. En primer lugar, se explicará la trama enviada del LabVIEW al microcontrolador y después, se detallará la enviada del microcontrolador al LabVIEW. Cada *byte* está representado en codificación binaria para una mejor comprensión de la codificación, detallando cada uno de ellos en la Figura 3.13.

Como ya se comentó el *byte* STX y el *byte* ETX corresponden a una codificación predefinida en ASCII. La codificación de los 3 *bytes* restantes, se justifica a continuación:

- El *byte* del código de acción incluye dos tipos de variables. Se decidió dividir el *byte* en 2 (parte alta y parte baja) para implementar en cada una de ellas un tipo de acción. En la parte alta está definido el sentido de movimiento de la vagoneta, como se lista en la Figura 3.13. En la parte baja se codificó la velocidad a la cual la vagoneta se mueve, siendo “velocidad 1” la más lenta y “velocidad 16” la más rápida. No se ha decidido ampliar el rango de velocidades, ya que en condiciones normales se trabajará a velocidad máxima. Por último, DATA HIGH y DATA LOW se encargan de llevar la posición a la que la vagoneta debe ir. Cuando el microcontrolador reciba los 2 *bytes*, pondrá uno a continuación del otro para interpretar correctamente la posición definida por el programa LabVIEW.

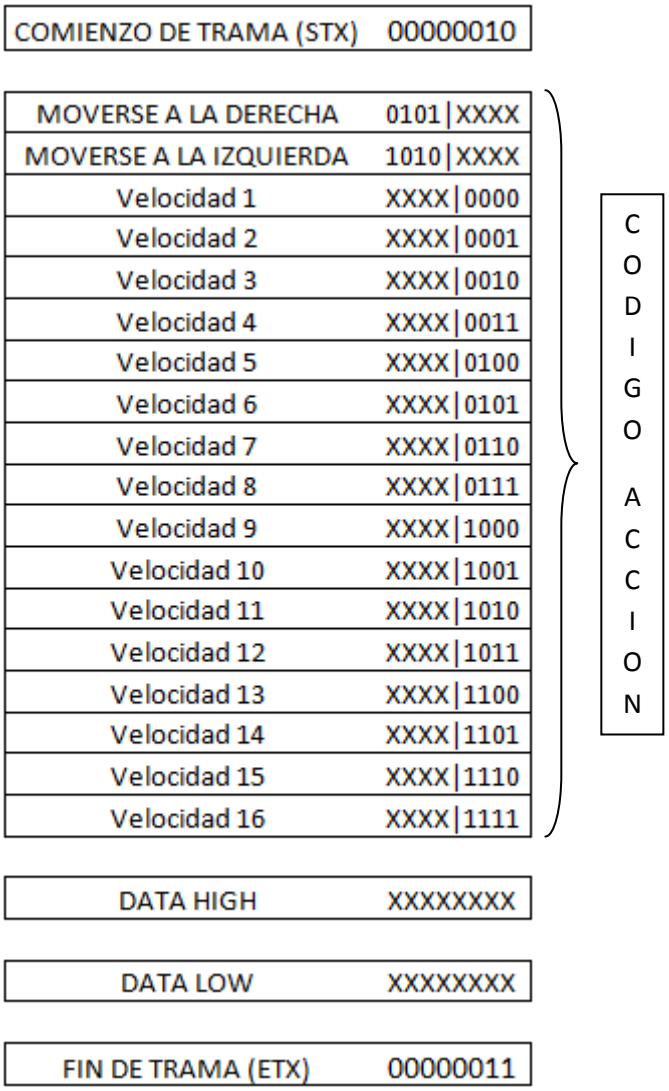


Figura 3.13 Codificación de la trama del LabVIEW para el microcontrolador

En la Figura 3.14, se muestra la codificación de la trama procedente del microcontrolador que el programa LabVIEW interpretará. Como se puede observar, la única diferencia que existe, es el tipo de código de acción que interpreta cada dispositivo, siendo el resto de *bytes* iguales. En este caso, el programa en LabVIEW distinguirá si los datos que le llegan a través de DATA HIGH y DATA LOW se refieren a la posición actual en la que se encuentra la vagoneta sobre el carril, o la posición en la que está el extremo derecho del carril.

COMIENZO DE TRAMA (STX)	00000010
POSICION ACTUAL	00000000
EXTREMO DERECHO	10110011
DATA HIGH	XXXXXXXX
DATA LOW	XXXXXXXX
FIN DE TRAMA (ETX)	00000011

Figura 3.14 Codificación de la trama del microcontrolador para el programa LabVIEW

3.3.3. BLOQUE DE INICIALIZACION DEL PUERTO SERIE

En este apartado se describe cómo inicializar el puerto serie en el programa LabVIEW [2]. Aunque se comentó que los dispositivos están conectados mediante un cable USB, en realidad la comunicación entre ambos es serie. Es decir, la conexión entre ambos se realiza con un cable USB, pero el tipo de protocolo de comunicación es serie. A lo largo del documento se hará mención a comunicación serie y USB indistintamente, teniendo para el actual documento el mismo significado (protocolo de comunicación serie). El bloque completo de inicialización se muestra en la Figura 3.15, la cual pasaremos a detallar a continuación.

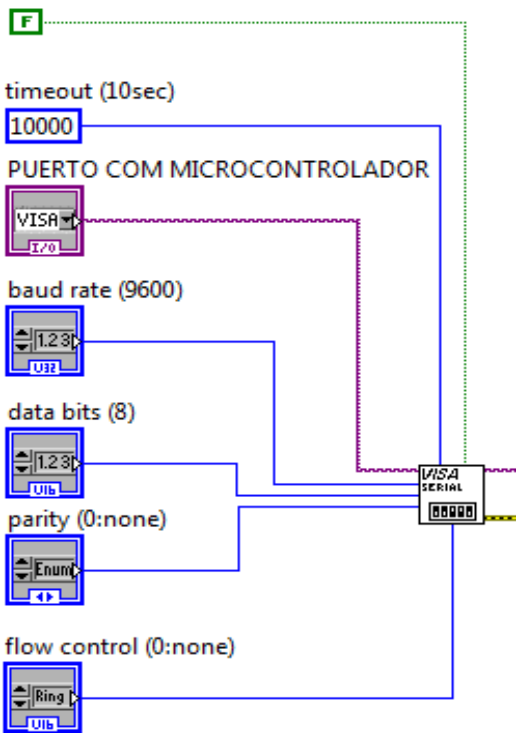


Figura 3.15 Parámetros de inicialización del puerto serie

Por una parte, se encuentra el bloque que aparece en color blanco en la parte derecha de la Figura 3.15 llamado “VISA SERIAL”. Este es el encargado de realizar una función específica consistente en abrir el puerto serie y establecer una comunicación con otro dispositivo. En la parte izquierda de la figura se muestran todos los parámetros configurados para que el bloque “VISA SERIAL” funcione correctamente. A continuación se describen detalladamente los parámetros de configuración de la comunicación listándolos desde el que aparece en la parte superior al que aparece en la parte inferior:

- En primer parámetro de configuración, es un valor booleano False (F). Esta variable advierte al bloque “VISA SERIAL” de que no ejecute la acción que tiene definida por defecto. Según ella, si recibe el valor hexadecimal 0x0A (equivalente a “\n” o salto de línea) al leer del puerto serie, los siguientes valores son interpretados como una trama diferente. Es decir, en la aplicación que nos ocupa, si al enviar los 5 *bytes* del protocolo de comunicación descrito anteriormente, en uno de ellos se envía el valor 0x0A, el proceso de lectura termina, y el resto de *bytes* son interpretados como si se hubiesen enviado a posteriori (En otra trama), dando lugar a un envío erróneo de información.

- El tiempo de espera (*timeout*) estipulado por el programa LabVIEW es, por defecto, de 10 segundos. Durante este tiempo el bloque “VISA SERIAL” está intentando abrir el puerto serie. Si en ese tiempo no consigue abrirlo, la aplicación mostrará un mensaje de error advirtiendo de lo ocurrido. Dicho mensaje es generado por el propio LabVIEW sin posibilidad de ser modificado.

- El bloque llamado “PUERTO COM MICROCONTROLADOR”, será el encargado de indicar al bloque “VISA SERIAL” qué puerto COM será el que se abra para realizar el intercambio de información. Dependiendo del ordenador o del puerto USB que se conecte el microposicionador, el puerto COM variará. Este parámetro es el primero que tiene que configurar el usuario final una vez que decida controlar el microposicionador con la aplicación desarrollada.

- El control de bit de datos (*data bits*), se ha estipulado en 8, dado que es el numero de bits más corriente a la hora de realizar cualquier comunicación. Hay que destacar que el programa LabVIEW solamente permite elegir en la programación de este parámetro entre 5 y 8 bits de datos. La velocidad en baudios (*baud rate*) que se ha estipulado, por defecto, para la comunicación es de 9600 baudios. Sin embargo, este no es un parámetro estático, es decir, se

ha programado la posibilidad de configurar la velocidad a la que se conectarán los dos dispositivos a gusto del usuario que maneje la aplicación.

- En último lugar, el control de flujo (*flow control*) y la paridad (*parity*) son nulos, ya que se definió así para el microcontrolador del proyecto desarrollado previo al que nos ocupa. Aunque el propio LabVIEW toma por defecto estos valores si no se conecta ningún control al bloque "VISA SERIAL", se ha decidido incluirlas en la programación por si en un futuro hiciesen falta.

3.3.4 BLOQUE DE ESCRITURA DEL PUERTO SERIE

En las siguientes líneas se describe el bloque de escritura del puerto serie [3]. En la descripción de este bloque y del resto (en los siguientes apartados), se ha evitado, intencionadamente, mostrar del diagrama de bloques completo de los instrumentos virtuales programados (en los anexos se adjuntan los más importantes), debido a su extensión y a la complejidad de las estructuras de código y conexiones. No obstante, el código completo, se incluirá en una copia en CD adjunta. Cada bloque funcional incluye a su vez otros bloques funcionales a los cuales se accede pinchando sobre los primeros. En este sentido, para explicar el detalle de programación se han seleccionado y extraído las funciones consideradas claves para la aplicación.

De las funciones que incluye el bloque de escritura del puerto serie, la más importante es propiamente la función que escribe en el puerto serie (ver Figura 3.16). Esta función escribe en el puerto COM, (COM1 ó COM2 ó COM3…) que se le indica a través de la línea morada. Este COM coincidirá con el configurado previamente en el bloque "VISA SERIAL" ya que ambos están conectados entre sí. Los parámetros a escribir, los recibe por la línea rosa, y la línea amarilla es la encargada de llevar los posibles mensajes de error que surjan.

Figura 3.16 Escritura en el puerto serie (VISA WRITE)

Los parámetros a escribir fluyen por la línea rosa, que debe su color al tipo de datos de que se trata. El color rosa indica que los datos que se enviarán por el bloque "VISA

WRITE” serán de tipo carácter (*string*). Este tipo de datos no resulta intuitivo para el usuario final a la hora de introducir posiciones o visualizarlas, por lo que se utilizará en primer lugar un tipo de dato numérico, más concretamente un dato numérico-decimal, que luego se transformará a *string*. LabVIEW no hace distinción entre datos decimales, binarios o hexadecimales a la hora de las conexiones entre bloques. Este tipo de conexión de datos se representa con una línea azul, tal como se muestra en la Figura 3.17. Cuando se quiera visualizar un tipo especifico de dato numérico, habrá que indicarlo en los propios terminales y no en las líneas de conexión.

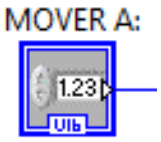


Figura 3.17 Ejemplo de representación de dato numérico

Cuando se trabaja con la posición actual de la vagoneta, surge un problema añadido a la hora de enviar la trama. Como ya se comentó anteriormente, se necesitan 2 *bytes* para especificar dónde se encuentra la vagoneta. Por tanto, para indicarle al microcontrolador dónde se desea que se posicione, se debe separar dicho número (por ejemplo, 2600) en dos *bytes* para poder interpretarlos correctamente en recepción. En LabVIEW existe una función que realiza esta tarea y se representa en la Figura 3.18 [4].

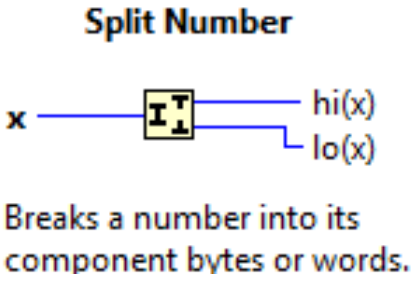


Figura 3.18 Función de LabVIEW para separar un *byte* en dos *bytes*

En ella se observa que el dato numérico que entra, es dividido en *bytes* o *words* a la salida. Esta función es transparente al tipo de dato numérico que entra, realizando correctamente su función en cualquier caso. En el caso que nos ocupa, los números decimales de entrada solamente tomarán valores correspondientes a *words* (suficientes para codificar las 5200 posiciones que tiene el carril), por tanto las salidas solo podrán contener *bytes*.

Para realizar la conversión de los datos que interactúan con el usuario (datos numéricos) en los datos que serán enviados a través de “VISA WRITE” (*strings*), se ha implementado una función que realiza dicha conversión, como se muestra en la Figura 3.19. Esta función llamada ”BUILD ARRAY” primero convierte el número en un *array* numérico, y posteriormente la función “BYTE ARRAY TO STRING” pasa del *array* numérico al *string*.

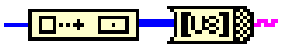


Figura 3.19 Conversión de datos numéricos a datos tipo *string*

Una vez realizada la conversión, los datos están listos para ser enviados al puerto serie si se desea. En particular para el diseño del proyecto, como se precisa enviar 5 *bytes* (referentes al protocolo de comunicación descrito), ó se realiza un bucle FOR para enviar cada uno de ellos, o se concatenan formando un solo *array*. En LabVIEW existe una función que realiza el concatenado de n *bytes* tal como se muestra en la Figura 3.20.

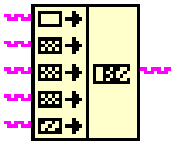


Figura 3.20 Concatena 5 *bytes* para generar la trama del protocolo de comunicación

Se observa que tiene 5 entradas referentes a los 5 *bytes* del protocolo, y una única salida que va conectada en la entrada del bloque “VISA WRITE”. En este punto habrá que ocuparse sólo de juntar los 5 *bytes*. La función “VISA WRITE” ya se encargará de enviarlos de forma secuencial como si se hubiera programado un bucle FOR, ya que en la inicialización se especificó que se enviaran en paquetes de 8 bits.

3.3.5 BLOQUE DE LECTURA DEL PUERTO SERIE

El bloque de lectura tiene las mismas características generales que el bloque de escritura. Al contrario que el bloque de escritura, este bloque está continuamente a la

espera de la recepción de los datos del puerto COM. Para realizar la lectura del puerto serie, se utiliza la función “VISA WRITE”, que se representa en la Figura 3.21.

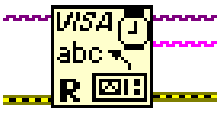


Figura 3.21 Lectura del puerto serie (VISA WRITE)

En ella se observa que se dispone de una conexión de color morado, correspondiente al puerto COM del que se leerá y otra de color amarillo que genera los mensajes de error. Como sucedía en el bloque de escritura, la línea de color rosa devolverá lo leído del puerto. Para poder visualizar estos datos (*string*) en formato decimal, primero se deben convertir. Esta conversión la realiza la función ”STRING TO BYTE ARRAY” representada en la Figura 3.22.

Figura 3.22 Conversión de datos tipo *string* a datos numéricos

Una vez realizada la conversión, se deben separar los *bytes* recibidos para comprobar si pertenecen a una trama del protocolo normalizado o, por el contrario, resultan ser erróneos. La conversión descrita se encarga únicamente de generar un tipo de dato numérico en forma de cadena o *array*, por lo que hubo que separar adicionalmente los *bytes* del *array* con la función “INDEX ARRAY” mostrada en la Figura 3.23. Notar que en la figura se ha incluido también el conversor de la Figura 3.22 para facilitar su comprensión. La función “INDEX ARRAY” extrae los *bytes* requeridos a través de los parámetros de la izquierda. Para la aplicación se ajusta perfectamente dividiendo el *array* en 5 tal como se muestra en la Figura 3.23.

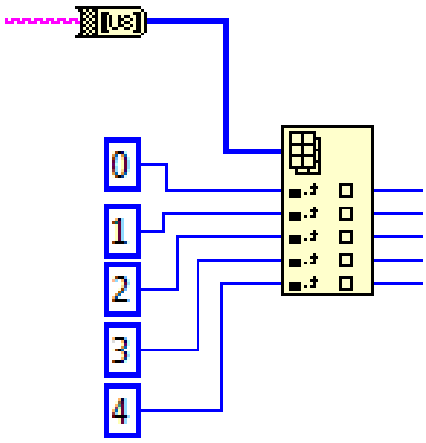


Figura 3.23 Función de LabVIEW para separar 5 *bytes* que se leen del puerto serie

Una vez se han separado los *bytes*, se comprueba que el inicio de trama y el final de trama coinciden con lo descrito en el protocolo de comunicación. Si no es así, se rechazan todos *bytes*. Por el contrario, si coinciden, se pasa a determinar qué código de acción (o de orden) se ha recibido para discriminar en qué variable se guardarán los *bytes* DATA_HIGH y DATA_LOW. Si se recibe el número 0b00000000, se guardarán en las siguientes variables globales: “DATA_HIGH” y “DATA_LOW” (ver Figura 3.24).

Destacar también, que el *byte* de código de acción siempre se guardará en una variable llamada “CODIGO DE ORDENES”, sea cual sea su valor (ver Figura 3.24).

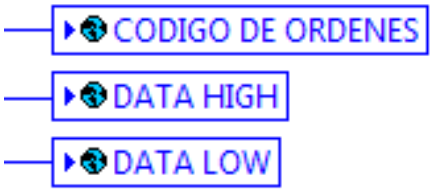


Figura 3.24 Variables globales

Por otro lado, si el código de acción coincide con el número binario 0b10110011, “DATA_HIGH” y “DATA_LOW”, pasarán a guardarse en otra variable global llamada “FIN CARRIL” (ver Figura 3.25).

Figura 3.25 Variable global FIN CARRIL

En la Figura 3.25 también aparece otro bloque funcional junto a la variable global. Este bloque es dual al comentado en la Figura 3.18, en él se produce la concatenación de los 2 *bytes* que entran por la parte izquierda, dando lugar a un solo elemento del tamaño de la suma de sus dos entradas [4]. (Ver Figura 3.26).

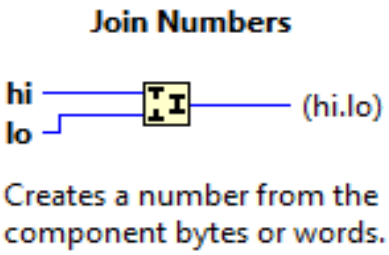


Figura 3.26 Función de LabVIEW para juntar 2 *bytes* en un byte

Se observa que la salida está formada, primeramente por el elemento superior, seguido del elemento inferior. En el caso particular que nos ocupa, el elemento “hi” será “DATA_HIGH”, y el elemento “lo” será “DATA_LOW”, dando lugar al número correspondiente al final del carril del sistema microposicionador.

3.3.6 BLOQUE DE EJECUCION DE BUCLE

Una vez cumplido el objetivo de realizar una interfaz de comunicación bidireccional entre el PC y el microcontrolador para el control del movimiento del sistema microposicionador, se amplió el sistema añadiendo una nueva funcionalidad. La tarea consistió en la programación de secuencias de movimiento del motor a modo de bucles repetitivos. El bloque que se describe en este apartado hace referencia a esta tarea. La necesidad de generar bucles de movimiento surge como consecuencia de los procesos propios de caracterización de dispositivos electroópticos. Son habituales, por ejemplo, algunos procesos de *test* como el sondeo de la homogeneidad superficial de las muestras o la medida de características específicas en distancias micrométricas. En estos casos, puede ser necesario un tiempo de retardo entre bucles de movimiento para que el usuario monitorice la muestra, tome datos, modifique las condiciones de contorno del problema o realice capturas de la célula *test*.

Para dar solución a esta demanda, el sistema que por el momento sólo conseguía posicionar la vagoneta en determinados puntos estratégicos, tuvo que ser ampliado. A partir de este punto, las tareas de posicionamiento automático y espera entre ciclos de movimiento se establecieron como configurables a través de unos parámetros de control. Es decir, con la simple introducción por el usuario de unos datos de control, el sistema se posicionaría en determinados puntos y, una vez allí, esperaría un tiempo determinado para realizar capturas, por ejemplo. La definición de dichos parámetros de control fue la siguiente:

1. Sentido de movimiento de la vagoneta: Con este parámetro de control se especifica el sentido de movimiento que llevará la vagoneta a la hora de realizar las capturas.

2. Número de posiciones a mover entre una captura y otra: En este caso, se le indica al microcontrolador cuántas posiciones tiene que realizar entre una captura y otra.

3. Número de ejecuciones de captura: Este parámetro indica el número de capturas que se quiere realizar y, en general, el número de tiempos de espera entre ciclos de movimiento.

4. Espera necesaria para realizar la captura: Es el tiempo de espera hasta el comienzo del siguiente ciclo de movimiento. Para el caso de la captura fotográfica de imágenes, el tiempo suficiente para que la cámara pueda realizar una captura y procesarla, llegado el caso.

Para realizar este proceso de manera autónoma y secuencial, se desarrollaron los bloques de programa representados en la Figura 3.27, que se detallan a continuación. Se observa que la tarea conlleva 4 bloques de trabajo, alguno de los cuales lleva implícita la comunicación entre LabVIEW y microcontrolador. En primera instancia, y dependiendo del sentido del movimiento, se procede a enviar la nueva posición en la que capturar una imagen al microposicionador (según el bloque de escritura del apartado 3.3.4). Una vez que se pone en movimiento la vagoneta, el programa LabVIEW se queda en espera hasta que surja un evento entrante (interrupción de lectura del buffer) en el puerto serie (segundo bloque de la Figura 3.27).

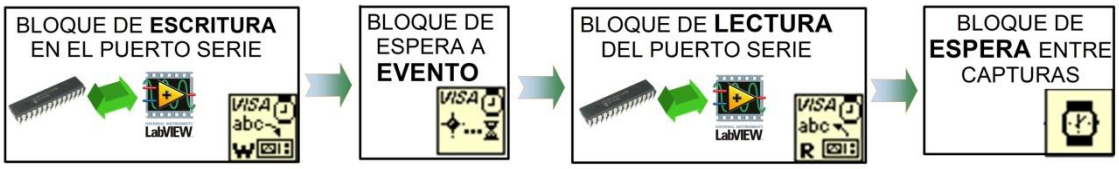


Figura 3.27 Cadena secuencial de funciones del bloque de ejecución de bucle

La función que realiza este elemento del diagrama de bloques se representa en LabVIEW con la función de la Figura 3.28. Se observa cómo el tipo de evento se especifica a través de un cuadrado azul; en este caso es un evento de tipo lectura del puerto serie. El resto de conexiones tienen el mismo significado que las ya descritas en el bloque de lectura y escritura.

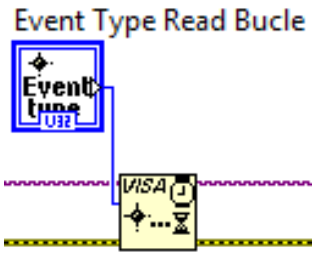


Figura 3.28 Función de espera hasta nuevo evento de lectura.

Una vez que la vagoneta ha llegado a su destino, el microcontrolador envía su posición nueva. Tras ello, el LabVIEW recibe el evento y pasa a ejecutarse el tercer elemento del diagrama de bloques, el bloque de lectura. Su funcionalidad es la misma que la detallada en el apartado 3.3.5. Una vez el programa LabVIEW interpreta los datos recibidos y conoce la posición de la vagoneta, espera un tiempo para realizar la captura, que viene determinado por los parámetros de control. La función de espera (cuarto elemento del diagrama) viene representada en LabVIEW con la Figura 3.29. En ella se observa que, a través del bloque azul (con el valor temporal deseado), se introduce en el bloque de la derecha el tiempo que debe esperar el programa para comenzar a ejecutar una nueva secuencia de movimiento. El bloque de la derecha es la función de espera, “WAIT (ms)” de LabVIEW.

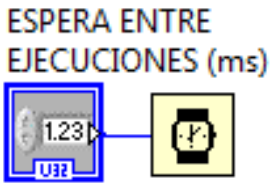


Figura 3.29 Función de espera entre secuencias de movimiento consecutivas

Una vez acabada la ejecución de la secuencia de elementos de la Figura 3.27, que como ya se ha comentado se realizan de una manera secuencial, se comprueba si el número de capturas o intervalos de tiempo de espera que se quieren realizar se ha completado. Si este número coincide con el número de ejecuciones que lleva el bucle, este parará. Si no es así, se seguirá ejecutando hasta llegar al número de veces indicado.

3.3.7 BLOQUE DE CIERRE DEL PUERTO SERIE

En este último bloque de programa desarrollado en LabVIEW, se comenta la función clave implementada para cerrar el puerto serie de una manera correcta. Dicha función representa una parte ínfima del total del código como ocurría con el bloque de inicialización del puerto serie. Esta función viene representada en la Figura 3.30, y sus dos subfunciones se comentan a continuación [3].

Figura 3.30 Diagrama de bloques para la ejecución del cierre del puerto serie

Se observa que está compuesto por dos funciones que en LabVIEW se llaman “VISA CLOSE” y “SIMPLE ERROR HANDLER” comenzando de izquierda a derecha. La primera simplemente cierra el puerto serie que se le indica por la línea morada, y la segunda muestra un mensaje de error (ver Figura 3.31) si alguna función que esté conectada a la línea amarilla ha tenido algún problema [3].

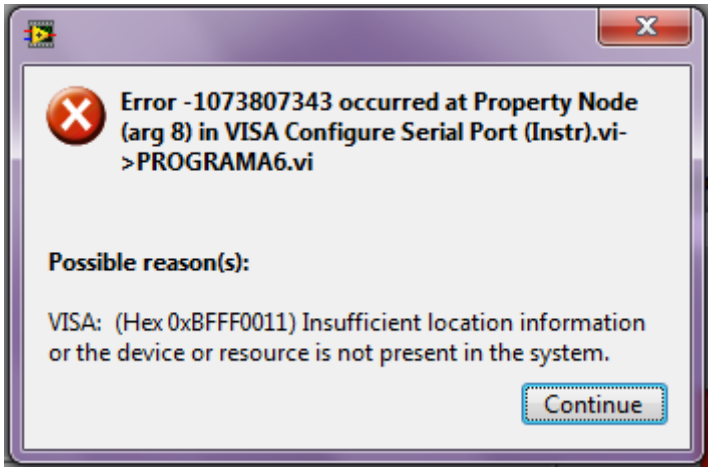


Figura 3.31 Mensaje de error de LabVIEW

Si por algún motivo el cierre de sesión del puerto no se hiciese, se estaría creando un conflicto con futuras ejecuciones de la interfaz. Al intentar inicializar un puerto ya abierto, se estarían dando valores a los mismos parámetros incurriendo en fallos del puerto.

CAPÍTULO 4

DISEÑO DE LA APLICACIÓN A NIVEL DE MICROCONTROLADOR

4.1 PROGRAMACIÓN DE LA APLICACIÓN EN LENGUAJE C

El capítulo anterior se dedicó a la explicación exhaustiva de la programación de la aplicación a más alto nivel, es decir, en entrono gráfico con el programa LabVIEW. Sin embargo, como ya se adelantó en ese capítulo, algunos de los bloques funcionales programados en LabVIEW se apoyan en llamadas a funciones desarrolladas también en lenguaje de alto nivel, pero que se han implementado en lenguaje C (en entorno no gráfico). La programación de estas funciones en C está íntimamente ligada al control directo sobre el dispositivo microcontrolador. Se trata de un microcontrolador PIC18F2550 que mueve el motor del sistema y ejecuta las órdenes de movimiento que se efectúan a través de los pulsadores. En el capítulo que nos ocupa se explicará detalladamente el diseño y la programación del código de control del sistema microposicionador *hardware* en lenguaje C.

Como se ha comentado en el capítulo de motivación y objetivos del proyecto, previamente a este trabajo, se implementó en un proyecto anterior [1] un código básico de control para el microposicionador basado en lenguaje C. El código que se muestra en este capítulo constituye un código mejorado al que allí se programó. Se han introducido múltiples cambios sobre el mismo, por lo que se ha considerado conveniente proceder a explicar el código completo haciendo especial hincapié en los aspectos de programación que suponen una mejora tanto de código, como de funcionalidad para el sistema final. En este sentido, en este capítulo se muestran los diagramas de flujo que se han realizado para todas las funciones explicando el código que ejecutará el microcontrolador.

Se recuerda que el sistema microposicionador completo está basado fundamentalmente de dos grandes bloques implementados: el bloque electrónico y el bloque mecánico (carril y vagoneta). Para el objetivo de programación que nos ocupa, se considerará el bloque electrónico, tomando como punto de partida la caja donde se encuentra insertada la electrónica de control y en particular el microcontrolador PIC18F2550. En la Figura 4.1, se muestra el detalle del panel frontal de la misma con los botones o pulsadores que activarán las ejecuciones del código cuando se trabaje en modo remoto. En la descripción de las funciones se hará mención a los pulsadores cuando se hable de la acción que lleva asociada cada uno de ellos.

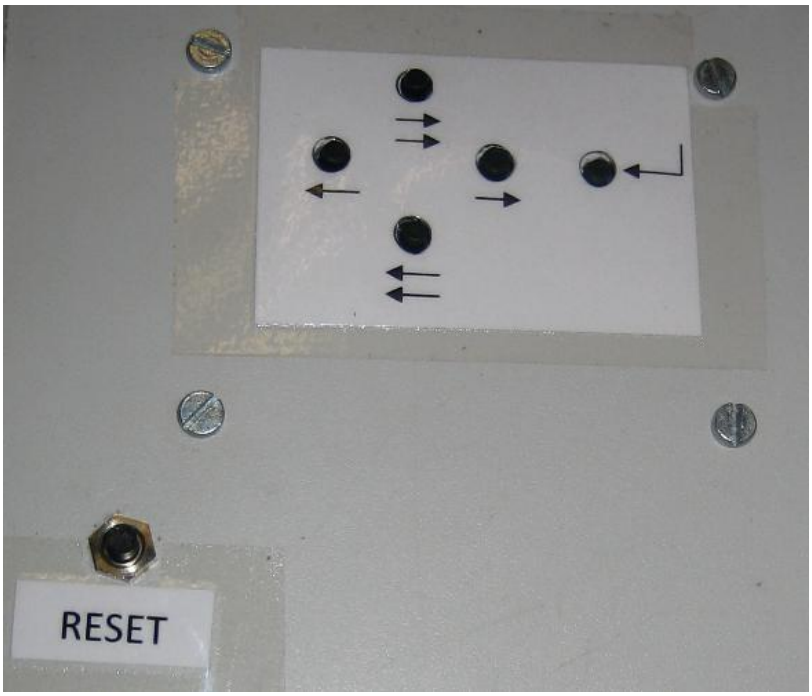


Figura 4.1 Detalle del panel frontal del dispositivo de control del sistema microposicionador. En él se ubica el microcontrolador PIC18F2550

4.2 ESTRUCTURA DEL CÓDIGO PROGRAMADO

El código implementado a nivel de microcontrolador se ha programado en lenguaje C. Sin embargo, para que la explicación del código sea más llevadera y sencilla de entender, a lo largo de los apartados siguientes de la memoria, se han presentado las funciones a través de los diagramas de flujo que llevan asociadas. El detalle de las líneas de código se puede consultar en el apartado de anexos de la memoria. Para explicar los diagramas de flujo se han utilizado 6 tipos de símbolos (ver Figura 4.2).

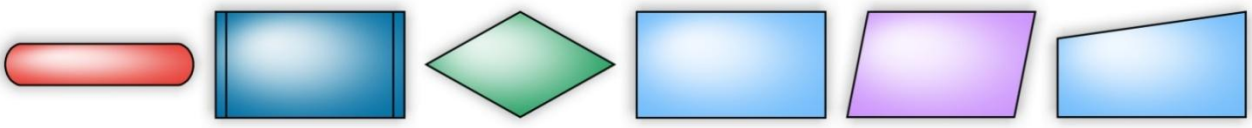


Figura 4.2 Simbología utilizada en los diagramas de flujo

El significado de cada símbolo es el siguiente:

- El primero (inicial), de color rojo, es el de inicio y fin de una función.
- El rectángulo de color azul oscuro, es una llamada a una función.

- El rombo verde, es un condicional.
- El rectángulo azul claro, es una instrucción.
- El rectángulo violeta, es un valor que devuelve una función.
- El último (final), de color azul, determina una ampliación de flujo.

Con la ayuda de esta simbología, las funciones que se han implementado se listan a continuación. Se han agrupado en varios apartados en función de la afinidad de las acciones asociadas a cada una de ellas.

- ✓ Función ”MAIN”. (Apartado 4.3)
- ✓ Funciones de configuración e inicialización: (Apartado 4.4)
  - ”CONFIGURACION”
  - ”TEST_LCD”
  - ”INICIALIZAR_CARRIL”
  - ”POSICION_DERECHA”
  - ”POSICION_IZQUIERDA”
- ✓ Funciones de movimiento de la vagoneta: (Apartado 4.5)
  - ”POSICION_ACTUAL”
  - ”IMPRIMIR_POSICION_ACTUAL”
  - “LEER_PULSADORES”
  - ”ACCIONES_MANUALES”
  - ”PASO_MOTOR”
  - ”GUARDA_EEPROM”
- ✓ Funciones de envío de información al LabVIEW: (Apartado 4.6)
  - “ENVIO_POSICION_ _LABVIEW”
  - ”ENVIO_FIN_CARRIL_LABVIEW”.
- ✓ Funciones del puerto USB: (Apartado 4.7)
  - ”USB_CONECTADO”
  - ”ACCIONES_USB”

4.3 FUNCIÓN “MAIN”

El diagrama de flujo de la función “MAIN” viene representado en la Figura 4.3. Se observa que comienza haciendo unas llamadas a las funciones “CONFIGURACION”, “LCD_INIT“ y ”TEST_LCD”, que son inicializaciones y configuraciones tanto de la LCD como de los puertos del microcontrolador.

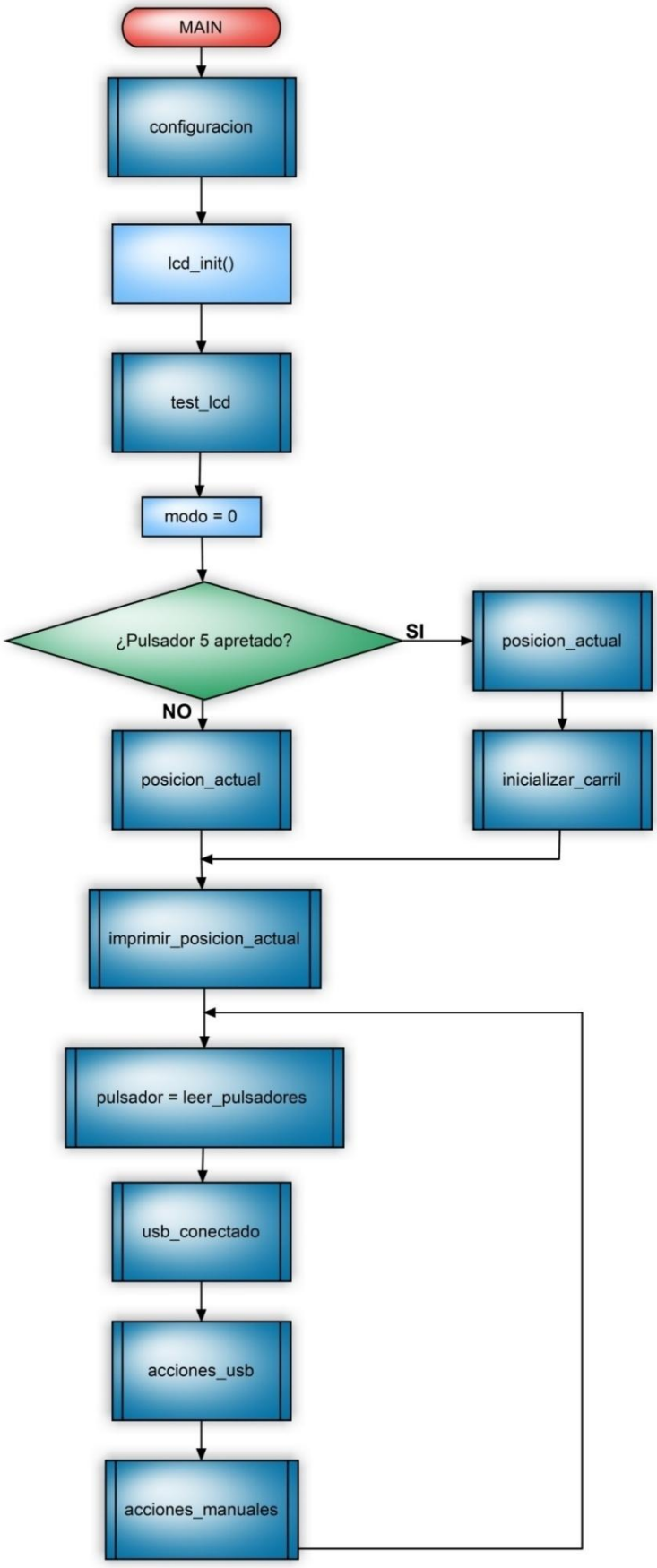


Figura 4.3 Diagrama de flujo de la función "MAIN"

Para testar el correcto funcionamiento de la LCD, “TEST_LCD” imprime un mensaje por pantalla. Tras inicializar el microcontrolador y la LCD, se guarda una variable llamada modo, en la que se indicará si el USB está conectado o no.

- Si modo = 0 se considera que el USB no está conectado. La variable se inicializa a 0 como se indica en el flujograma del microcontrolador y podrá cambiar su estado en la función “USB_CONECTADO”.
- Si modo = 1, está conectado.

Una vez hechas las inicializaciones correspondientes, el microcontrolador comprueba si el pulsador 5 está pulsado (Tras realizar un reset, si se deja pulsado el botón 5, se le indica al microcontrolador que se quiere realizar una inicialización del carril, siendo dicho pulsador el correspondiente con el símbolo ↵ mostrado en la Figura 4.1). Si es así, llamará a la función “POSICION_ACTUAL”, que es la que se encarga de leer los datos de la EEPROM, y posteriormente llamará a “INICIALIZACION_CARRIL”, que hará un calibrado para determinar los extremos del carril. “POSICION_ACTUAL” y “INICIALIZACION_CARRIL”, se detallarán más adelante. Si no se pulsa el botón 5, el microcontrolador solamente llamará a la función “POSICION_ACTUAL”, sin realizar la inicialización del carril.

Cuando se haya ejecutado una u otra acción dependiendo de si se pulsa el pulsador 5, se llamará a la función “IMPRIMIR_POSICION_ACTUAL”, la cual imprimirá por la LCD la posición última en la que había quedado la vagoneta antes de haber apagado el sistema, o tras haber realizado un RESET.

En este punto la función “MAIN” entra en un bucle infinito, esto es, hasta que no se apague el sistema o se haga un RESET, lo único que hace son llamadas secuenciales a las siguientes funciones:

- “LEER_PULSADORES”
- “USB_CONECTADO”
- ”ACCIONES_USB”
- ”ACCIONES_MANUALES”

En primer lugar llama a “LEER_PULSADORES” para comprobar si se ha pulsado algún botón. Esa acción indicará que el usuario del sistema desea mover la vagoneta, por ejemplo. El valor que devuelve la función se guarda en la variable llamada “pulsador” para posteriormente determinar qué acción ejecutar. Una vez se comprueba el estado de los pulsadores, se comprueba si se ha conectado el USB [6] con la

función “USB_CONECTADO”. Finalmente, se llama a las funciones ”ACCIONES_USB” y ”ACCIONES_MANUALES”, que se encargan, la primera de interpretar las acciones procedentes del LabVIEW y la segunda de mover la vagoneta en el sentido oportuno si se ha pulsado algún botón o se lo ha indicado el programa desarrollado.

4.4 FUNCIONES DE CONFIGURACIÓN E INICIALIZACIÓN

En este apartado se incluyen los diagramas de flujo de las funciones “CONFIGURACION”, “TEST_LCD” e INICIALIZAR CARRIL”. Estas funciones están todas relacionadas con la configuración e inicialización del sistema, tanto de variables de trabajo como de estados de los dispositivos del sistema.

4.4.1 FUNCIÓN “CONFIGURACION”

En la Figura 4.4 viene representado el diagrama de flujo de la función “CONFIGURACION”. En esta función se inicializan todos los puertos del microcontrolador, ya sean temporizadores, contadores, conversores ADC, comunicaciones del puerto serie, etc.

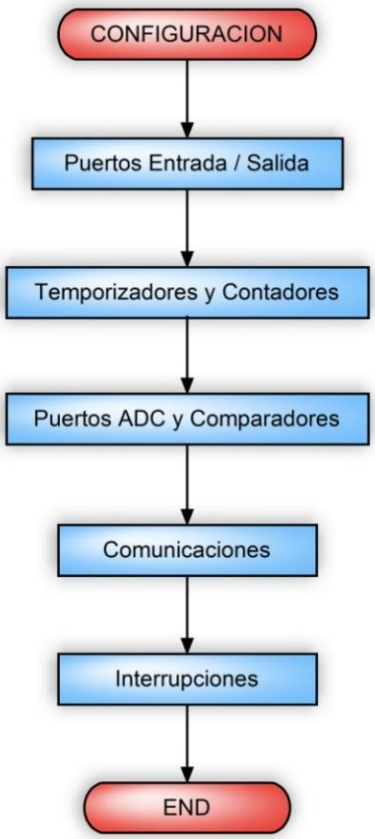


Figura 4.4 Diagrama de flujo de la función “CONFIGURACION”

Para el proyecto en desarrollo, solamente se han activado algunas de las funciones que se comentan a continuación:

1. Puertos de entrada y salida: Se ha asignado a cada uno de los pines del microcontrolador su configuración en función de si se trata de puertos de entrada o de salida.
2. Temporizadores y contadores: En este caso no se ha inicializado ningún temporizador o contador porque no son necesarios en la aplicación.
3. Puertos ADC y comparadores: Aquí se configura el ADC(0), y el ADC(1). Que son los referentes al USB y a los pulsadores. El resto se utilizan para otras aplicaciones.
4. Comunicaciones: En este caso solamente se configuran los parámetros necesarios para la comunicación serie tales como el control de flujo, los bits de datos, etc.
5. Interrupciones: Para la aplicación no ha sido necesario crear ninguna interrupción, por lo tanto estarán desactivadas.

4.4.2 FUNCIÓN “TEST_LCD”

La función “TEST_LCD” ha sido diseñada para verificar que la pantalla LCD funciona correctamente cuando arrancamos el microposicionador. El sistema cuenta con esta pantalla fundamentalmente para visualizar la posición de la vagoneta en todo momento. En la Figura 4.5, se muestra el diagrama de flujo de dicha función.

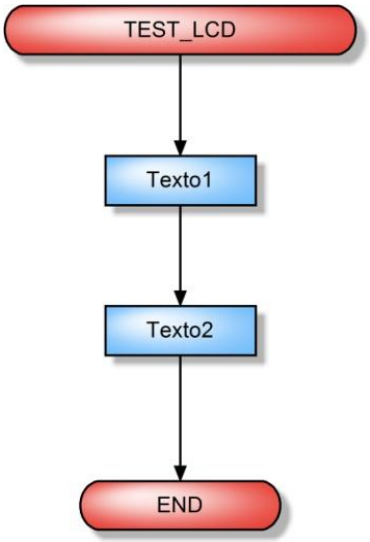


Figura 4.5 Diagrama de flujo de la función “TEST_LCD”

En él se puede observar que se imprimen dos textos. Dado que la LCD es del tipo 16x2 (16 columnas y 2 filas), cada texto sirve para verificar cada una de las filas, tal y como se muestra en la Figura 4.6.

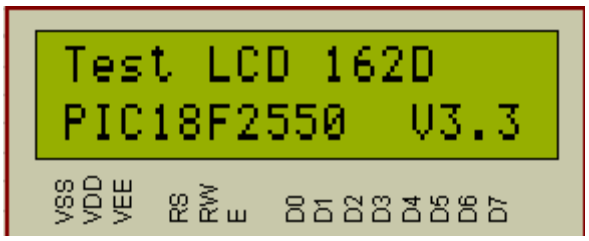


Figura 4.6 Verificación de funcionamiento de la pantalla LCD

4.4.3 FUNCIÓN “INICIALIZAR CARRIL”

En este apartado, se detalla el flujograma de la función “INICIALIZAR_CARRIL” que se representa en la Figura 4.8. Esta función tiene como objetivo general encontrar los finales de carrera, siguiendo siempre un orden, y guardarlos para su futura utilización.

El orden que seguirá es el siguiente:

1. Se envía a la vagoneta (esté donde esté) a buscar el “final de carril” izquierdo.
2. Una vez encontrado, se guarda en la EEPROM el valor cero que corresponde al inicio del carril.
3. Se vuelve a mandar a la vagoneta, pero esta vez hacia la derecha para encontrar el otro “final de carril” (derecho).
4. Una vez encontrado el otro extremo, se calcula la mitad del carril y se vuelve a enviar la vagoneta a la posición indicada.
5. Una vez allí, se muestra por la LCD el “final de carril” derecho para que el usuario conozca cuál es. Después, se envía toda la información al programa LabVIEW si estuviese conectado (se envía la posición actual y el final de carril). En la Figura 4.7 se observa el mensaje que aparece en la pantalla LCD sobre la posición del extremo derecho del carril para informar al usuario.

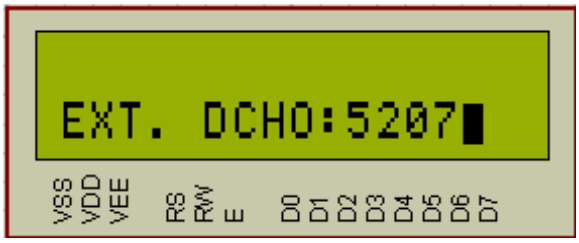


Figura 4.7 Mensaje en la pantalla LCD sobre la posición del extremo derecho del carril

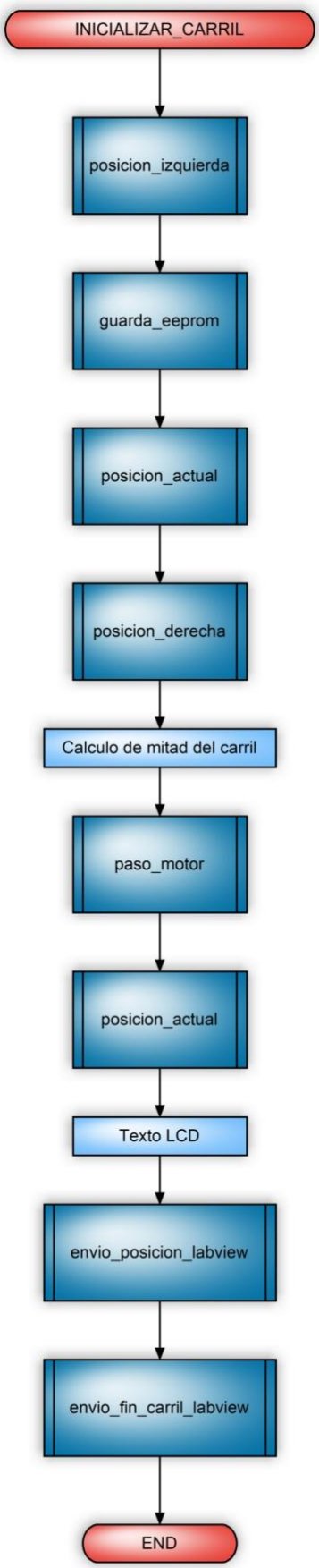


Figura 4.8 Diagrama de flujo de la función "INICIALIZAR_CARRIL"

4.4.4 FUNCIÓN “POSICION_DERECHA”

En este apartado se explica la función “POSICION_DERECHA” y su diagrama de flujo se representa en la Figura 4.9. Esta función ha sido implementada para ejecutar una parte de la inicialización del carril. Mecánicamente, en las proximidades de los extremos del carril se han colocado unos pulsadores, llamados “finales de carrera”, que permitan sensar básicamente si la vagoneta está cerca de dicha posición.

En primer lugar, se llama constantemente a la función “PASO_MOTOR” (encargada de mover la vagoneta) hasta que la vagoneta toca el “final de carril” derecho. Una vez que esto sucede, la posición actual “aux2” que en ese momento es el “final de carril” derecho, se iguala a otra variable que albergará dicha posición “extremo_derecho”. Por último, se llama a la función “GUARDA_EEPROM” y se guarda en una dirección de memoria diferente a posición actual, el valor del extremo derecho del carril.

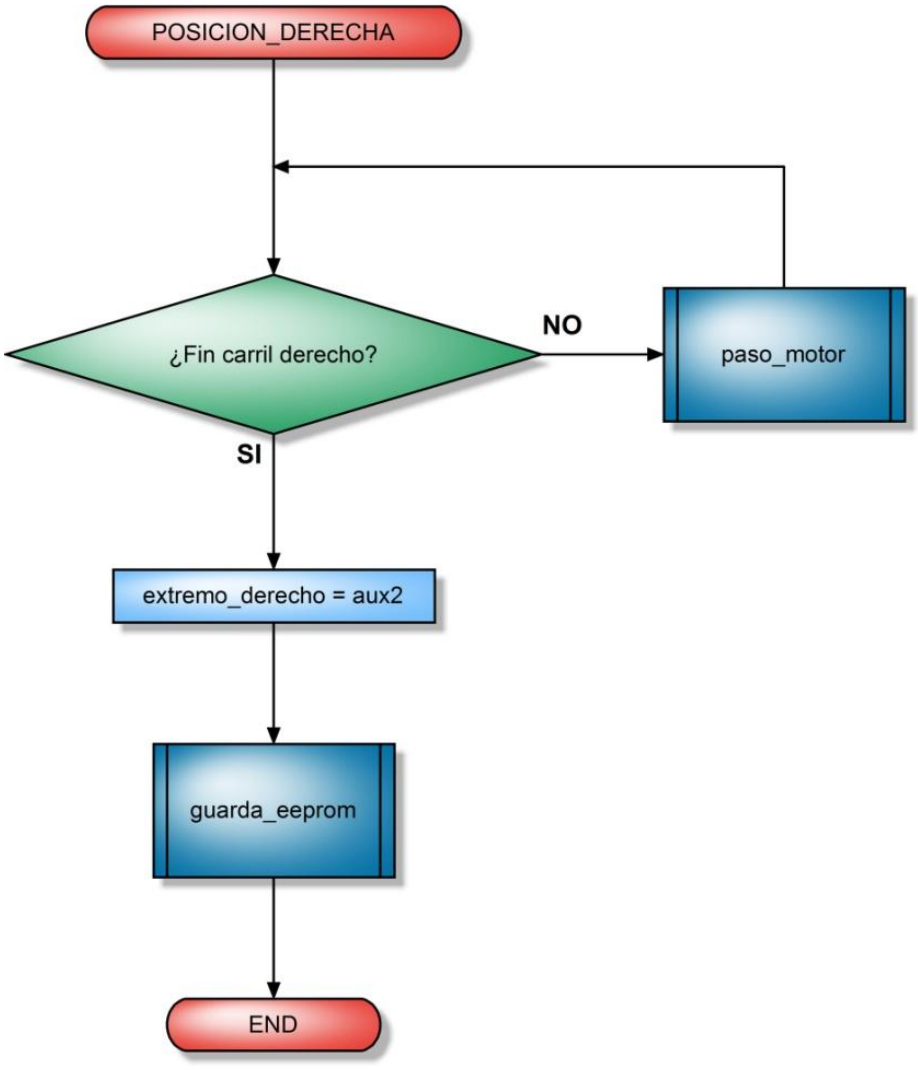


Figura 4.9 Diagrama de flujo de la función “POSICION_DERECHA”

4.4.5 FUNCIÓN “POSICION_IZQUIERDA”

En la Figura 4.10 se muestra el flujograma de la función “POSICION_IZQUIERDA”. Esta función es similar a la función “POSICION_DERECHA” descrita anteriormente. En este caso, se llama a la función “PASO_MOTOR” y se envía la vagoneta hacia la izquierda hasta tocar el “final de carril”. Una vez tocado, dicha posición se establece como el 0 del sistema de referencia completo. Es decir, todas las posiciones sucesivas estarán referidas al extremo izquierdo.

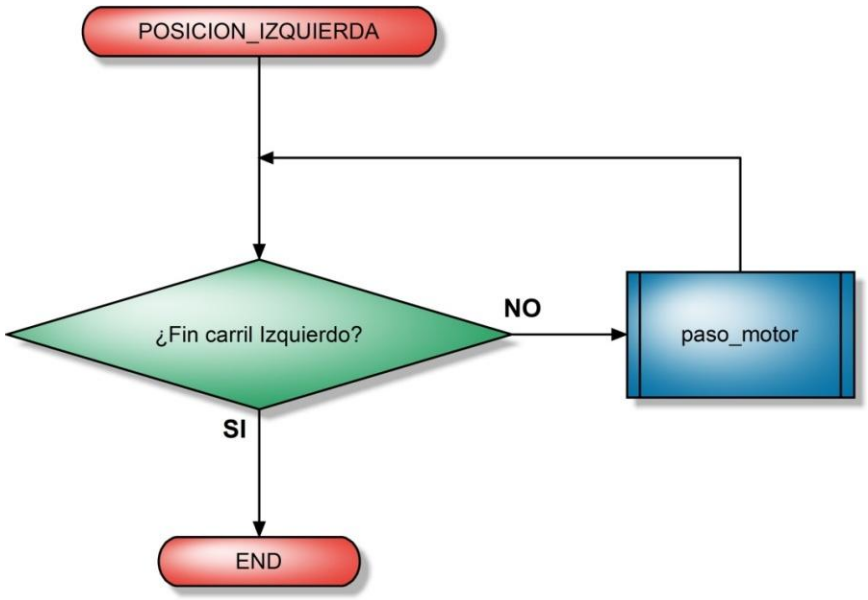


Figura 4.10 Diagrama de flujo de la función “POSICION_IZQUIERDA”

4.5 FUNCIONES DE MOVIMIENTO DE LA VAGONETA

En el apartado que se describe a continuación, se han incluido las funciones que están relacionadas en mayor o menor medida con el movimiento de la vagoneta. Las funciones se listan debajo:

- ”POSICION_ACTUAL”
- ”IMPRIMIR_POSICION_ACTUAL”
- “LEER_PULSADORES”
- ”ACCIONES_MANUALES”
- ”PASO_MOTOR”
- ”GUARDA_EEPROM”
- ”POSICION_DERECHA”
- ”POSICION_IZQUIERDA”

4.5.1 FUNCIÓN “POSICION_ACTUAL”

En la Figura 4.11 se muestra el flujograma de la función “POSICION_ACTUAL”. Esta función es la encargada de leer de la memoria interna del microcontrolador (EEPROM) los datos referentes a la posición actual de la vagoneta y del extremo derecho del carril. En la memoria EEPROM sólo es posible guardar datos de una longitud de 1 *byte*. Esto es un problema ya que los datos que utiliza el usuario final son números decimales de 4 cifras que, realizando la conversión a código binario, se corresponden con 2 *bytes*. Por ello, primero se lee el *byte* de mayor peso (direcciones altas) y se guarda en una variable de 2 *bytes*. Cuando se guarda dicho valor en la variable, no se puede especificar si se hace en la parte alta o en la baja. Cuando se igualan dos variables de diferente longitud, los datos se guardan en las direcciones bajas y las direcciones altas se hacen cero. Al ocurrir esto, se debe rotar lo guardado en la parte baja de la variable a la parte alta, ya que ésa es su posición definitiva.

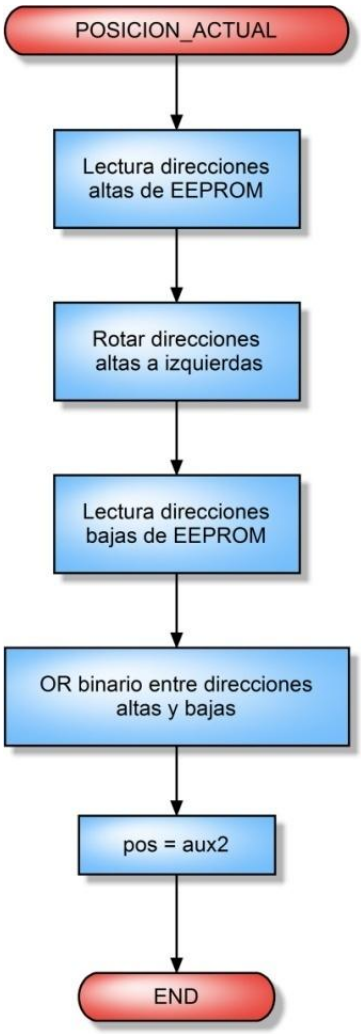


Figura 4.11 Diagrama de flujo de la función “POSICION_ACTUAL”

A continuación, se vuelve a leer de la EEPROM, pero esta vez las direcciones bajas. En esta ocasión no se puede igualar lo leído a la variable de 2 *bytes*, ya que sobrescribiría la parte alta como ya se ha comentado. Para esta ocasión, se debe realizar una suma binaria entre lo leído y la variable de 2 *bytes* (que en digital se corresponde con una operación OR). Una vez obtenida la posición actual, se realiza el mismo proceso para leer el "final de carril".

4.5.2 FUNCIÓN "IMPRIMIR_POSICION_ACTUAL"

En la Figura 4.12 se muestra el diagrama de flujo de la función "IMPRIMIR_POSICION_ACTUAL". Esta función realiza la impresión y actualización por la pantalla LCD de la posición de la vagoneta en el momento en que es llamada. La inclusión de este código en una función supone una mejora ya que la programación de esta parte del código de forma modular facilita su uso en diferentes partes del programa y evita refrescos innecesarios que conlleven una mala visualización de la pantalla LCD. En la ejecución del programa, solamente se refrescará la LCD cuando la vagoneta se mueva o cuando se arranque el microposicionador.

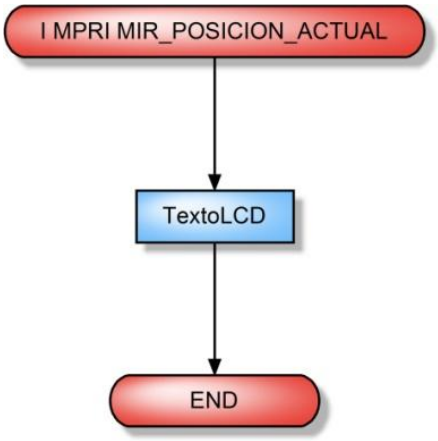


Figura 4.12 Diagrama de flujo de la función "IMPRIMIR_POSICION_ACTUAL"

En la Figura 4.13 se muestra un ejemplo del uso de la función que actualiza la posición de la vagoneta en la pantalla LCD. En el caso del ejemplo, la posición actual de la vagoneta es la 80. Todas y cada una de las posiciones que se representarán en la LCD y las que se enviarán a través del puerto USB, estarán siempre referenciadas al extremo izquierdo del carril (Posición 0).

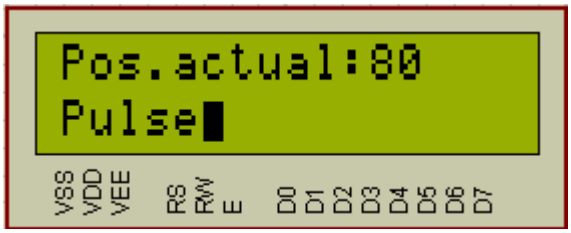


Figura 4.13 Mensaje en la pantalla LCD sobre la posición actual de la vagoneta

4.5.3 FUNCIÓN “LEER_PULSADORES”

En este apartado se muestra el diagrama de flujo de la función “LEER_PULSADORES”. Su implementación se representa en la Figura 4.14. Esta función está destinada a la lectura de los pulsadores del sistema microposicionador. Cada pulsador reflejará en la entrada del conversor analógico-digital una determinada tensión. Esta tensión capturada por el ADC (integrado en el microcontrolador), será posteriormente interpretada para llevar a cabo la acción correspondiente en función del botón pulsado. La función devolverá un número del 1 al 5, ya que existen 5 pulsadores y cada botón lleva asociado un número. La función “LEER_PULSADORES” se llama continuamente en el programa principal ya que se debe estar pendiente en todo momento del estado de los mismos.

La función implementada incluye alguna mejora respecto de su versión previa. La función inicial incluía un estado inicial por defecto cuyo objetivo era mostrar por la pantalla LCD la posición en la que se encontraba la vagoneta cuando ningún botón era pulsado.

Como ya se comentó en la función “MAIN”, esta función es llamada cíclicamente. Esto es así, porque el microcontrolador debe verificar el estado de los pulsadores por si alguno es accionado. Esta comprobación debe realizarse a una frecuencia lo suficientemente alta para que el microcontrolador no pierda en ningún caso la acción de una tecla pulsada. Para evitar que la pantalla LCD se refresque a una frecuencia visible por el ojo humano y su lectura sea demasiado incómoda, la orden de actualizar la pantalla LCD en cada comprobación de los pulsadores se extrajo del bucle cíclico de ejecución.

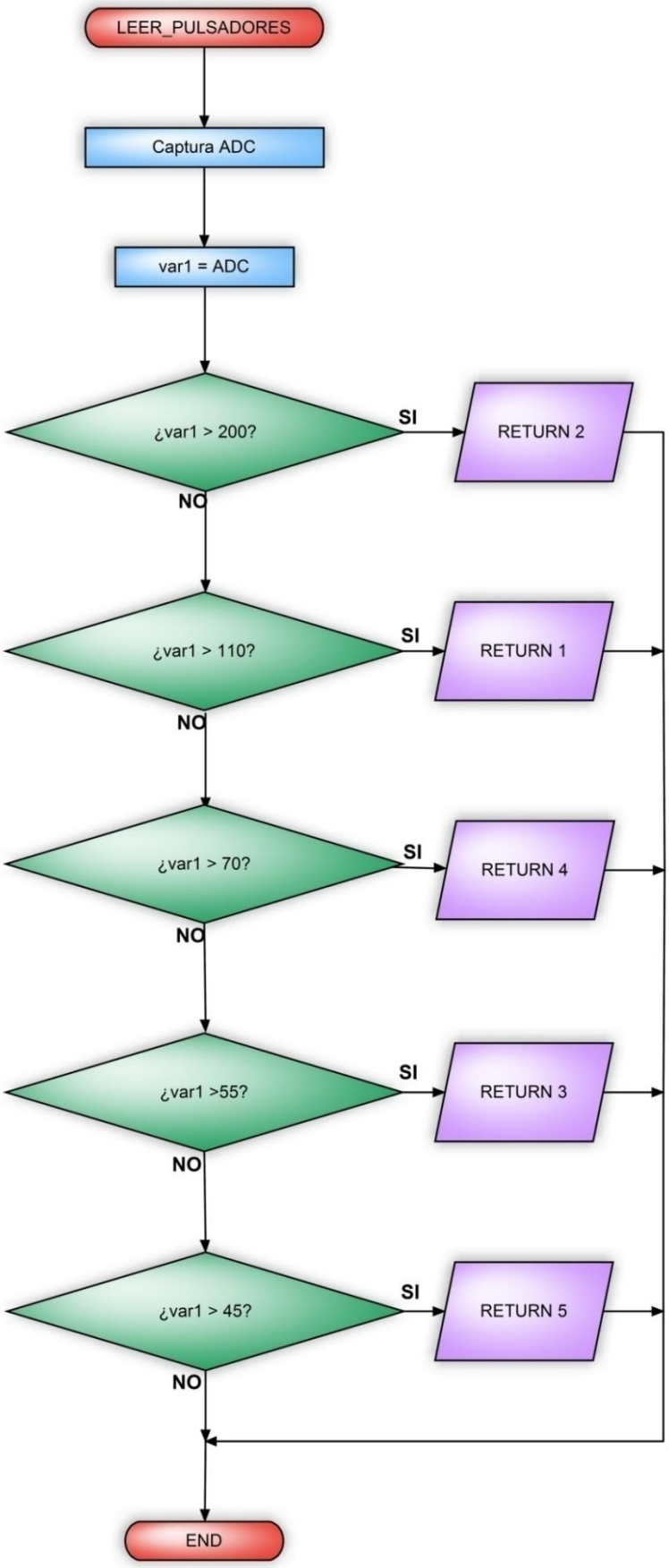


Figura 4.14 Diagrama de flujo de la función "LEER_PULSADORES"

4.5.4 FUNCIÓN “ACCIONES_MANUALES”

A continuación, se describe la función “ACCIONES_MANUALES” que está muy vinculada con la función “LEER_PULSADORES”. En la Figura 4.15, podemos observar dicha función reducida, la cual ejecutará una acción u otra dependiendo del valor recibido de “LEER_PULSADORES”.

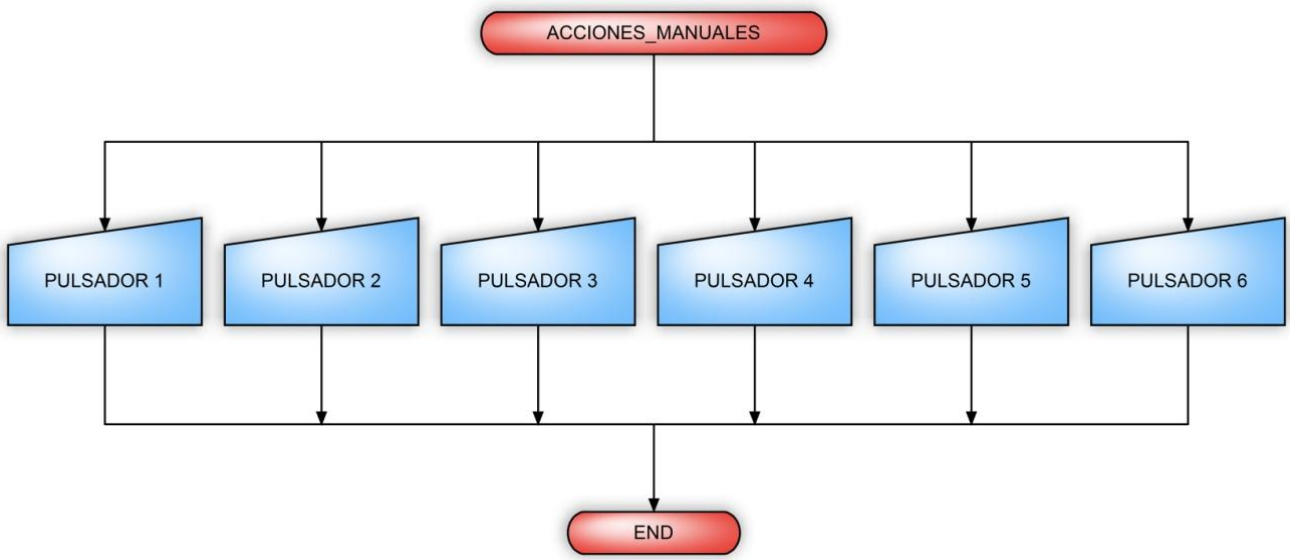


Figura 4.15 Diagrama de flujo de la función “ACCIONES_MANUALES”

Las acciones que realiza cada pulsador se han dividido debido a su extensión y serán explicadas por separado. Antes de detallar el funcionamiento de cualquier pulsador, conviene aclarar que se ha llamado “aux2” a la posición real en la que se encuentra la vagoneta, y “pos” a la posición de destino deseada.

PULSADOR 1

El pulsador 1 se refiere al incremento de la variable “pos” (ver Figura 4.16). Es decir, la posición futura (derecha) a la que se quiere enviar la vagoneta se incrementará de 5 en 5 cuando se pulse. El pulsador 1 se corresponde con el pulsador físico con el símbolo $\rightarrow$ mostrado en el panel frontal del controlador (retomar la Figura 4.1).

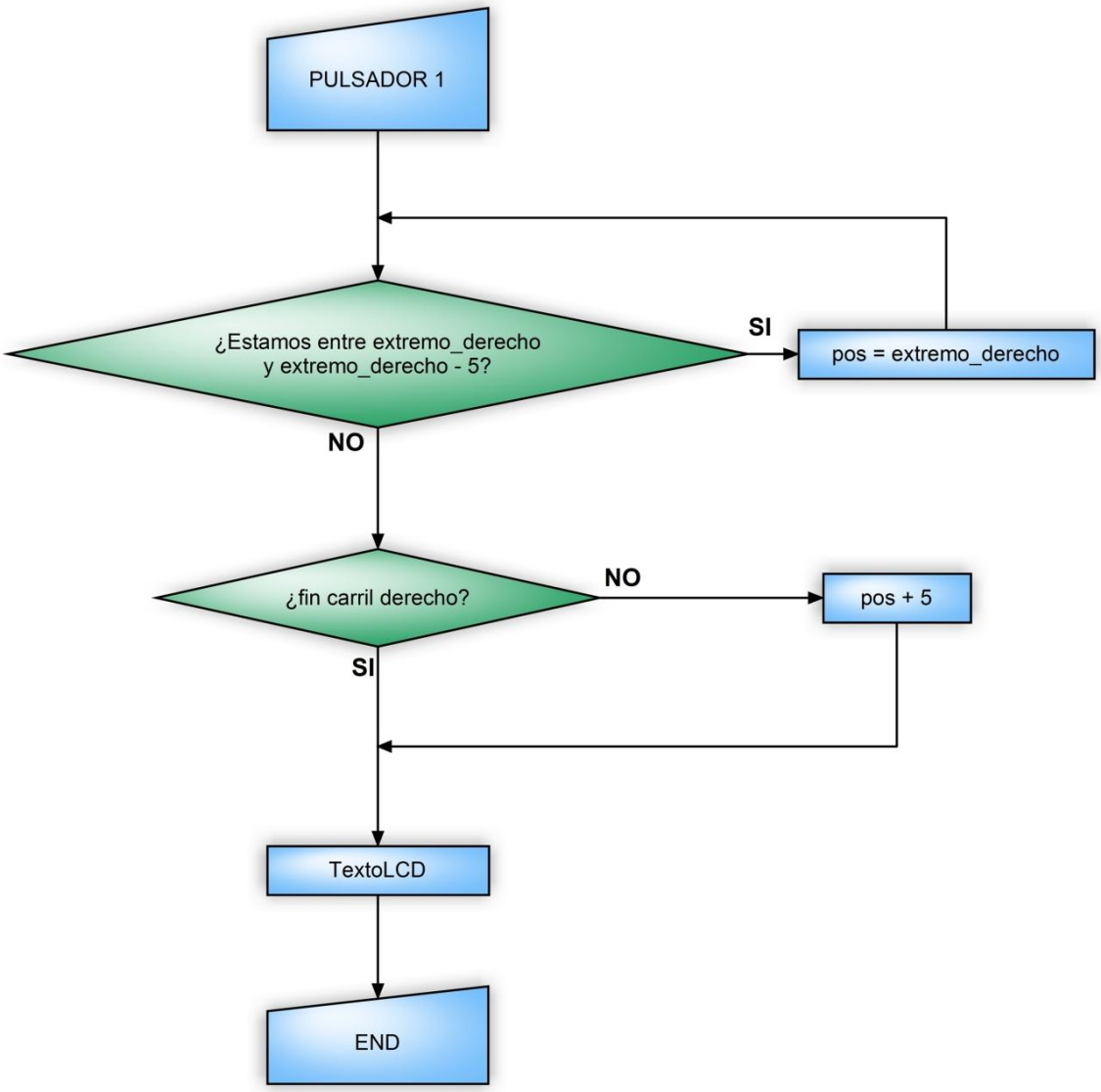


Figura 4.16 Diagrama para la ampliación de flujo "PULSADOR 1" asociado a la función "LEER_PULSADORES"

Primero comprueba si se está en el rango de las últimas 5 posiciones del carril en el extremo derecho. Si esto sucede se iguala "pos" a "extremo derecho". Si no se ejecuta este código, al sumar 5 posiciones a "pos", la vagoneta se saldría por del extremo derecho del carril, invalidando el funcionamiento del sistema.

Una vez comprobado que no se está en el rango anteriormente descrito, se pasa a comprobar si se ocupa la posición extrema derecha del carril, o no. Si es así no se suman más valores a "pos"; por el contrario, si no es así, se suman 5 posiciones. Tras realizar la acción de movimiento requerida, se muestra por la pantalla LCD la posición nueva a la que se desea enviar la vagoneta.

A modo de ejemplo se muestra en la Figura 4.17 el detalle del mensaje sobre la LCD en el que se indica la nueva posición a la que se quiere enviar la vagoneta. En este caso la posición actual es la 2658 y la posición futura la 2893.

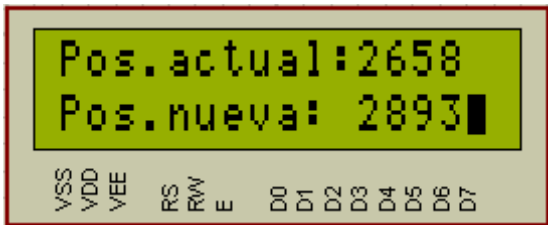


Figura 4.17 Mensaje en la pantalla LCD sobre la posición nueva a mover (derecha)

PULSADOR 2

El pulsador 2 será el encargado de mover la vagoneta hacia el lado izquierdo con recorridos cortos (de paso en paso). El símbolo asociado al botón físico que realiza esta tarea, ← , se muestra en el panel frontal del controlador de la Figura 4.1. El diagrama para la ampliación de flujo de PULSADOR 2 se desarrolla en la Figura 4.18.

Primeramente, antes de realizar ninguna acción, se comprueba si el “final de carril“ izquierdo está pulsado. Si no está pulsado el “final de carril” izquierdo se pasa a comprobar si la variable “aux2” (posición real en la que se encuentra la vagoneta) es mayor o igual que cero. Las dos condiciones anteriores puede que no sean iguales, es decir, verificar que el “final de carril” está presionado, no equivale a decir que la vagoneta se encuentre en la posición cero del carril. En determinadas situaciones críticas, el usuario puede presionar intencionadamente el “final de carril” izquierdo y provocar con ello la detención automática de la vagoneta como medida de protección o seguridad para los componentes del sistema electroóptico de caracterización.

Por tanto, en cualquiera de las dos situaciones, si se ha llegado al “final de carril” izquierdo o “aux2” es cero, se mostrará en la pantalla LCD el mensaje que aparece en la Figura 4.19.

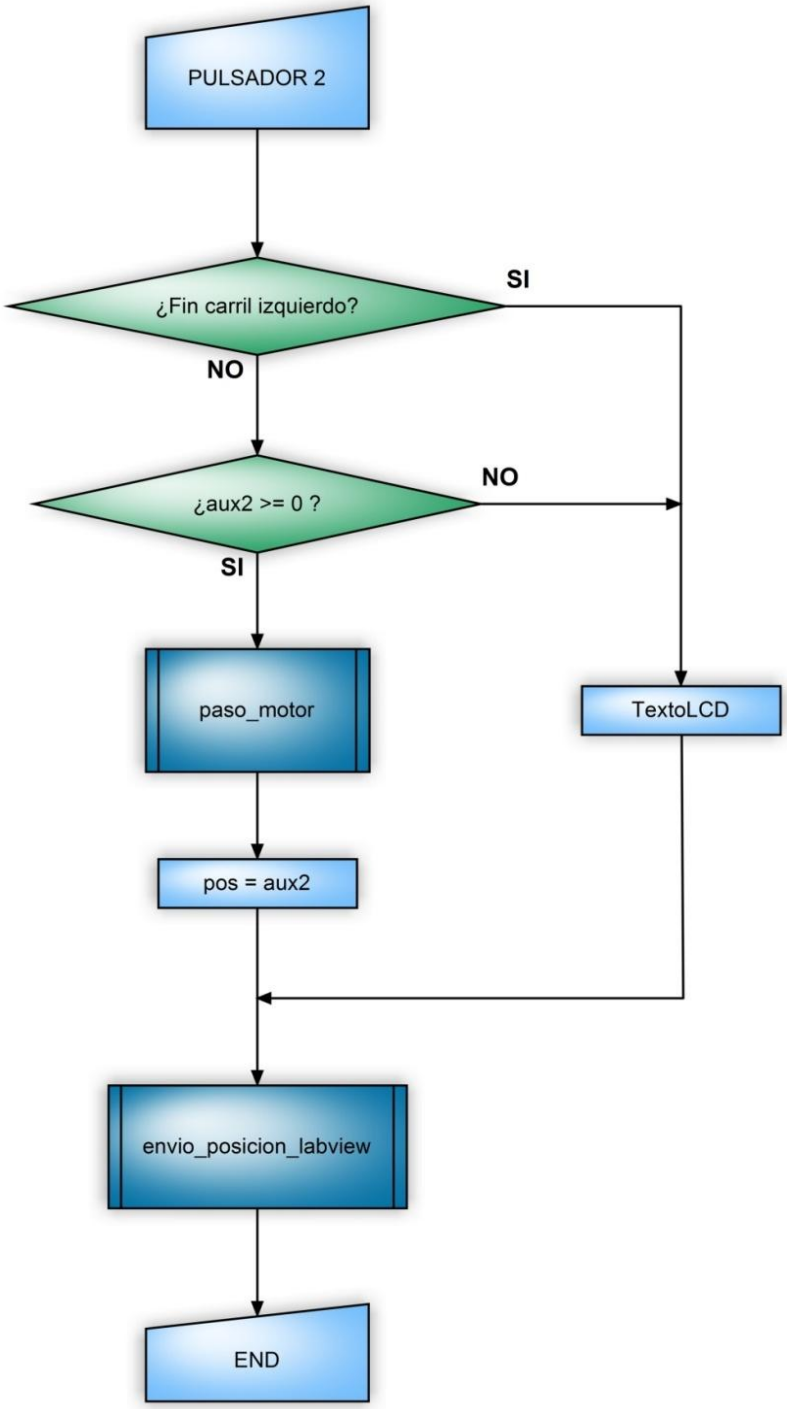


Figura 4.18 Diagrama para la ampliación de flujo "PULSADOR 2" asociado a la función "LEER_PULSADORES"

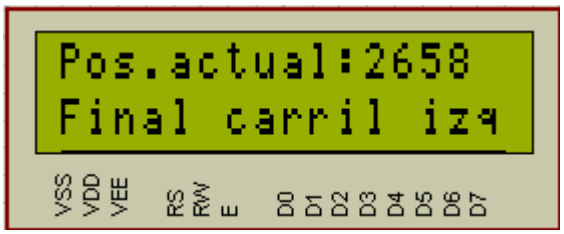


Figura 4.19 Mensaje en la pantalla LCD indicando que se ha llegado al "final de carril" izquierdo

Si no se está en ninguna de las dos condiciones anteriores, se llamará a la función "PASO_MOTOR", encargada de mover la vagoneta. Una vez la vagoneta llega a su destino, se igualan las variables "aux2" y "pos" con lo que la posición futura y la actual coinciden, y se envía al programa LabVIEW (si éste está conectado) la posición actual de la vagoneta.

PULSADOR 3

El pulsador 3, es el encargado de realizar la misma acción que el pulsador 2, pero en este caso para movimientos en sentido contrario (derecha) y su diagrama de flujo es el mostrado en la Figura 4.20. Y al igual que el caso anterior, la vagoneta se moverá de paso en paso. El símbolo del botón físico, del panel frontal del controlador, asociado al pulsador 3 se muestra en la Figura 4.1 como →.

En primera instancia, se pasa a comprobar si se ha llegado al "final de carril" derecho y "aux2" es mayor que la posición guardada del carril derecho. De forma complementaria a como ocurría con el pulsador 2, si no se cumple alguna de estas condiciones, se pasa a mostrar por la pantalla LCD el mensaje de "final de carril derecho" mostrado en la Figura 4.21.

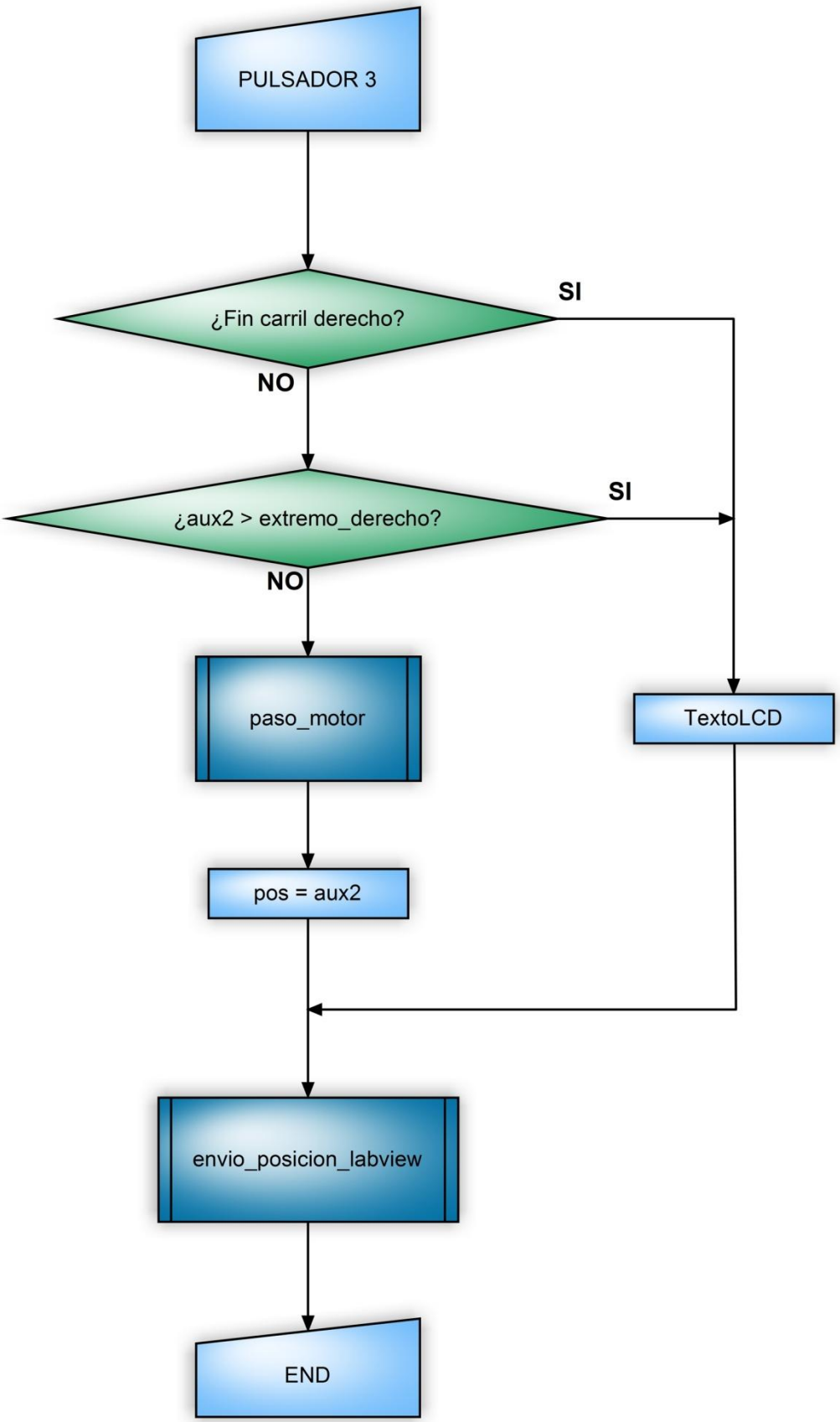


Figura 4.20 Diagrama para la ampliación de flujo "PULSADOR 3" asociado a la función "LEER_PULSADORES".

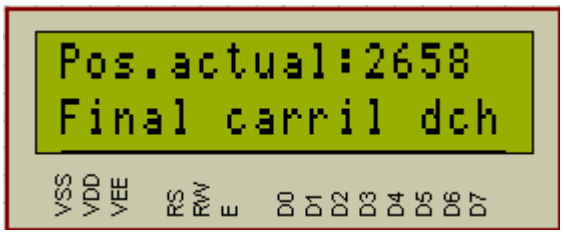


Figura 4.21 Mensaje en la pantalla LCD indicando que se ha llegado al "final de carril" derecho

En cambio, si estas condiciones se dan, se llamará a la función "PASO_MOTOR" para mover un paso la vagoneta, y posteriormente se igualarán las variables "pos" y "aux2" para asegurar que la posición actual y la futura coinciden. Una vez que la vagoneta ha ido a la posición requerida, se envía al programa LabVIEW (si éste está conectado), la nueva posición en la que se encuentra la vagoneta.

PULSADOR 4

El flujograma de la función del pulsador 4 se representa en la Figura 4.22. El símbolo del botón físico sobre el panel frontal del controlador se representa por ⇇, como se muestra en la Figura 4.1. Al igual que ocurría con el pulsador 3 (relacionado con el 2), el pulsador 4 está muy vinculado con el pulsador 1, es decir, se comprueban los mismos parámetros, pero en este caso los del extremo izquierdo.

Al avanzar a posiciones futuras de 5 en 5 pasos, lo primero que se comprueba es si la vagoneta se encuentra en el rango de 0 a 5 pasos desde el extremo izquierdo del carril. Si la vagoneta está en dicho rango y se acciona de nuevo el pulsador 4 se indica que se desea ir más pasos a la izquierda, por lo que habrá que igualar la posición futura a 0, que es el extremo izquierdo del carril. En caso contrario, se pasará a comprobar si por algún motivo el sensor de "final de carril" izquierdo está pulsado. Si se encontrase pulsado, la nueva posición de destino no cambiaría por motivos de seguridad, alertando de que puede haber un objeto obstaculizando el carril. En cambio, si no se está pulsando el "final de carril", se restarán 5 pasos a la nueva posición de destino. Una vez realizados estos pasos, se muestra por la pantalla LCD la posición de destino seleccionada para que el usuario final siga realizando las acciones que estime oportunas.

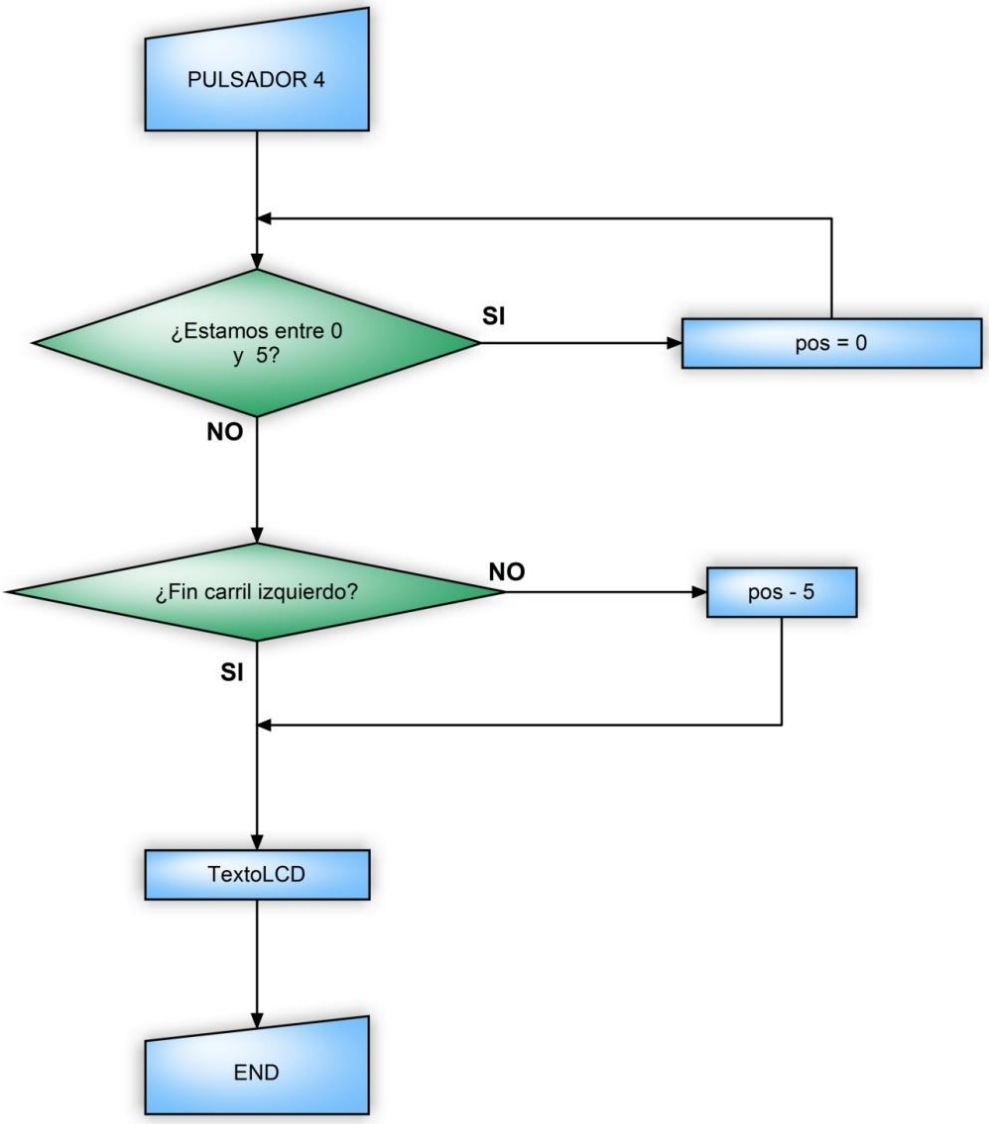


Figura 4.22 Diagrama para la ampliación de flujo "PULSADOR 4" asociado a la función "LEER_PULSADORES"

En la Figura 4.23 se muestra, a modo de ejemplo, el detalle de la pantalla LCD para el caso en que se desee que la vagoneta de desplace hacia la izquierda a una determinada posición. En el ejemplo la posición actual es la 2657 y la futura la 2032.

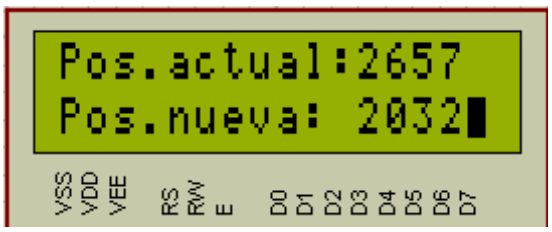


Figura 4.23 Mensaje en la pantalla LCD sobre la posición nueva a mover (izquierda)

PULSADOR 5

El pulsador 5 está asociado con el botón físico que aparece sobre el panel frontal del controlador con el símbolo ↵, como se muestra en la Figura 4.1. Su flujograma viene representado en la Figura 4.24 y, a continuación, se detallan sus acciones. Este botón es el encargado de hacer mover la vagoneta a la posición indicada por "pos" (destino deseado). Esto significa que los pulsadores 1 y 4 establecen la nueva posición, y el pulsador 5 es el encargado de ejecutar la acción de movimiento.

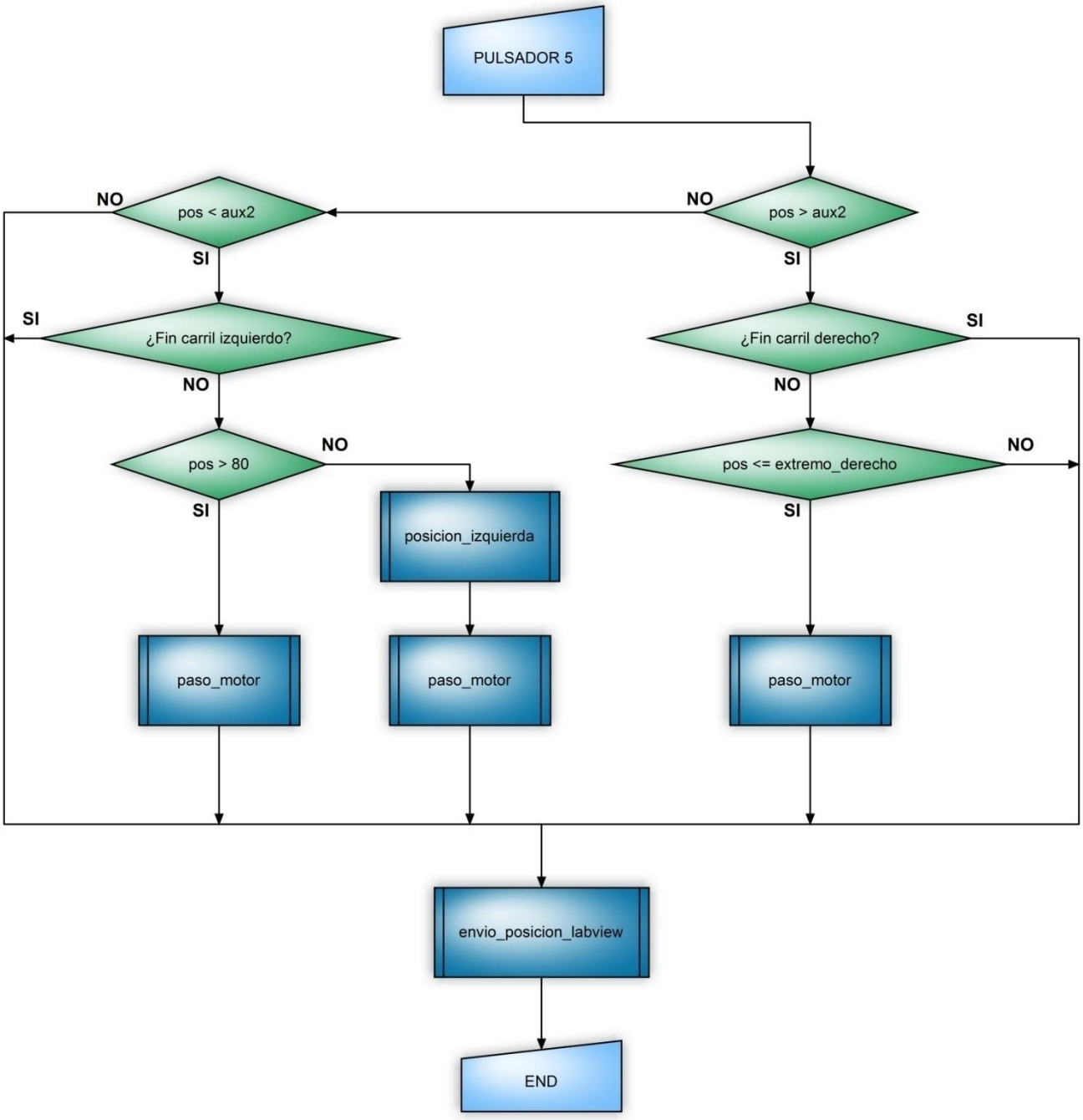


Figura 4.24 Diagrama para la ampliación de flujo "PULSADOR 5" asociado a la función "LEER_PULSADORES".

En un primer momento, se procede a discriminar si la vagoneta se moverá hacia la derecha o hacia la izquierda.

Si la posición de destino “pos” es mayor que la posición actual “aux2”, se procede a verificar que el “final de carril” derecho no está pulsado y que la posición de destino es menor que el valor de dicho extremo. Si esta condición se cumple, se llamará a la función “PASO_MOTOR” que será la encargada de mover la vagoneta. Por el contrario si alguna de las dos condiciones anteriores no se cumple, la vagoneta por seguridad no se moverá.

Por otro lado, si la posición destino “pos” es menor que “aux2”, en primer lugar se comprobará si el “final de carril” izquierdo se ha pulsado o no. Si se ha pulsado, la vagoneta no se moverá. Pero, por el contrario, si no ha sido pulsado, se verificará si “pos” es mayor de 80 pasos, o no. El motivo de estos 80 pasos, se expone a continuación. Para el sistema mecánico de transmisión que transforma la rotación del eje del motor en traslación de la vagoneta sobre el carril se utilizó una correa dentada. En la implementación mecánica del sistema que se llevó a cabo en un proyecto previo [1], hubo que decidir el sentido en el que estaría siempre tensada la correa, eligiéndose el derecho. De este modo, cada vez que la vagoneta se mueve hacia la derecha, no se realiza ninguna acción secundaria al propio movimiento de la misma. Sin embargo, cuando se quiere mover la vagoneta hacia la izquierda, se necesita realizar un reajuste de 80 pasos para retensar la correa y no perder la referencia de posición absoluta. Por ejemplo, si la posición inicial de la vagoneta fuese la 2000 y la posición a la que se desea enviar (futura) fuese la 1500 (Movimiento hacia el extremo izquierdo), primero se enviará la vagoneta a la posición 1420 (1500 - 80), y posteriormente irá a la posición 1500.

Una vez que se sabe que la posición destino es mayor de 80 pasos, se llamará a la función “PASO_MOTOR” y se procederá a mover la vagoneta con el reajuste pertinente. Si por el contrario la posición destino es menor de 80 pasos, primero se moverá la vagoneta a la posición 0, y posteriormente se desplazará a la posición indicada (entre 0 y 80 pasos).

Tras mover la vagoneta a la posición destino, se enviará al programa LabVIEW la posición actual de la vagoneta si es que existe comunicación entre los 2 dispositivos.

PULSADOR 6

El diagrama de flujo del pulsador 6 se detalla en la Figura 4.25. Este pulsador es el único que no está relacionado con la función “LEER_PULSADORES” puesto que ha sido diseñado exclusivamente para realizar la inicialización del carril cuando el programa LabVIEW se lo indica. En el flujo de ejecución primero se llama a la función “INICIALIZAR_CARRIL” encargada de realizar los pasos pertinentes para hallar los finales de carrera y enviarlos al programa LabVIEW. Posteriormente se llama a la función “IMPRIMIR_POSICION_ACTUAL”, que será la encargada de mostrar la nueva posición de la vagoneta.

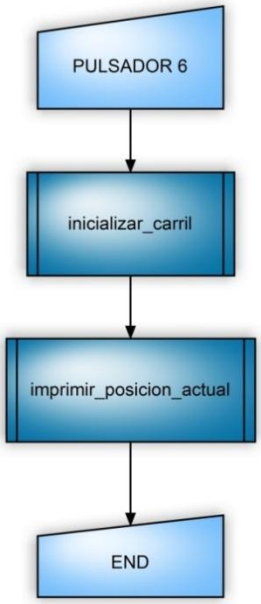


Figura 4.25 Diagrama para la ampliación de flujo “PULSADOR 6” asociado al programa LabVIEW

4.5.5 FUNCIÓN “PASO_MOTOR”

En la Figura 4.26 se muestra el diagrama de flujo de la función “PASO_MOTOR”. Esta función es la encargada de mover la vagoneta mecánicamente, en un sentido u otro, a través de un motor paso a paso y una correa. A esta función se le asignan 3 parámetros por referencia:

1. Sentido de movimiento del motor: ‘L’ (left, izquierda) ó ‘R’ (right, derecha).
2. Velocidad de movimiento. Velocidad inversamente proporcional al tiempo. La velocidad máxima se produce para un tiempo de 5 milisegundos, que es el tiempo entre pasos del motor.
3. Numero (n) de pasos que se moverá el motor.

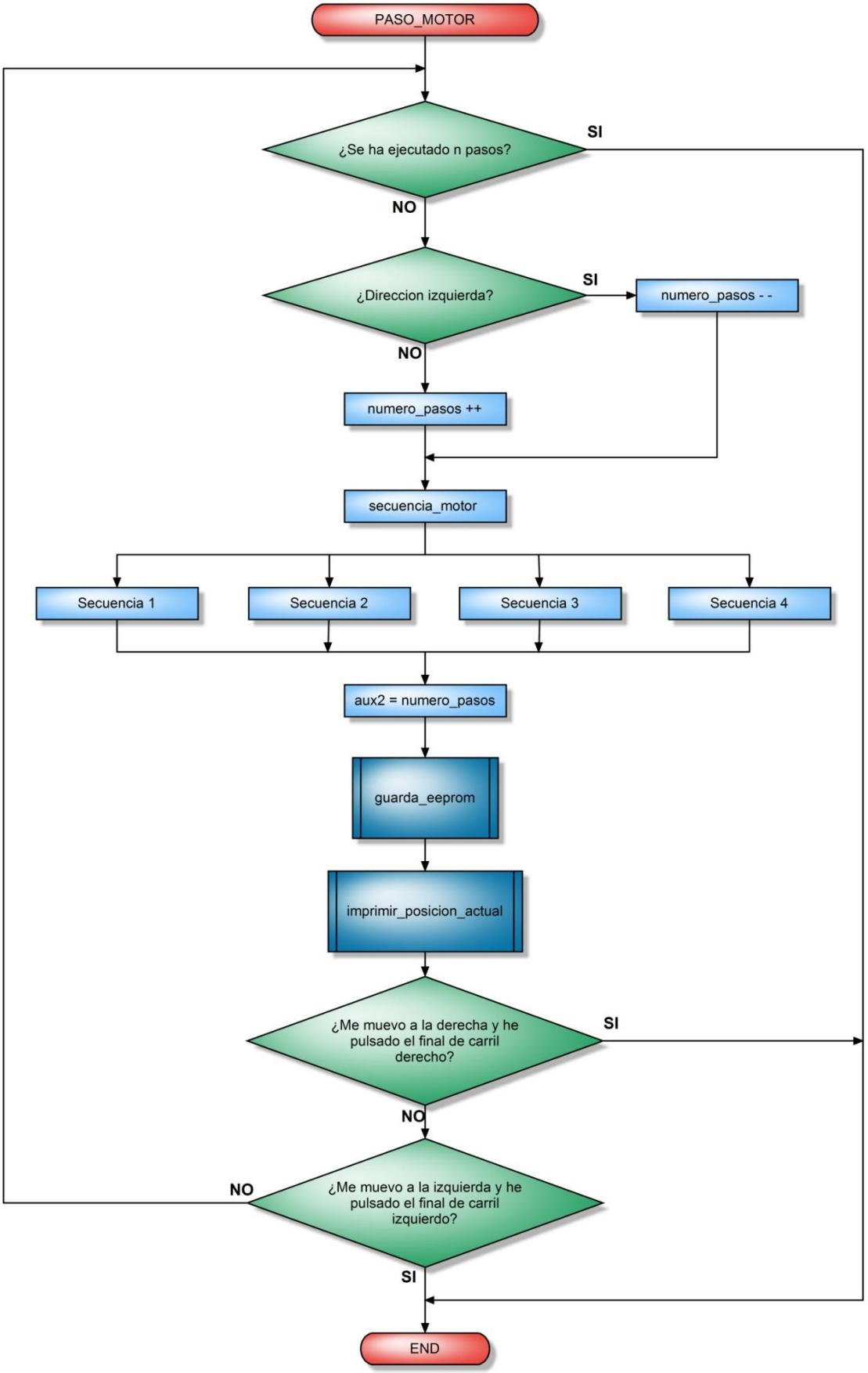


Figura 4.26 Diagrama de flujo de la función "PASO_MOTOR"

En primer lugar, se comprueba si el número de pasos que se deben ejecutar, coincide con el número de pasos que se han dado hasta el momento. La función se ejecutará hasta que coincidan ambos. Posteriormente, se verifica el sentido de movimiento de la vagoneta. Puesto que dependiendo de él, la "secuencia_motor" se llevará a cabo de forma ascendente o descendente. Es decir, para movimiento del motor hacia la derecha, la "secuencia_motor" se ejecutará siguiendo de forma cíclica la sucesión de instrucciones:

Secuencia 1, Secuencia 2, Secuencia 3, Secuencia 4, Secuencia 1, Secuencia 2…

Para movimiento del motor hacia la izquierda, la "secuencia_motor" se ejecutará siguiendo de forma cíclica la sucesión de instrucciones:

Secuencia 4, Secuencia 3, Secuencia 2, Secuencia 1, Secuencia 4, Secuencia 3…

Cada vez que se ejecuta una secuencia, se guarda en la memoria EEPROM la posición actual de la vagoneta y se muestra en la pantalla LCD. El motivo de guardar en cada paso la posición, viene impuesto por la posibilidad de que se produzca un RESET en cualquier momento. Si se guarda la posición de la vagoneta al finalizar la función (en vez de en cada paso), y el usuario final reinicia el sistema, surgirá un conflicto entre la posición que se guardó y la posición real de la vagoneta en el instante del RESET.

Además de guardar constantemente la posición actual en la EEPROM, se verifica también que, mientras se está moviendo la vagoneta, no se toca ningún "final de carril". Es decir, que para cada sentido de movimiento, su "final de carril" correspondiente no ha sido pulsado. Esta última parte, constituye una mejora introducida en el sistema que cubre una necesidad de seguridad del instrumental óptico que se utiliza para la caracterización, como se ha adelantado en apartados previos. Es decir, anteriormente, sólo se podía parar el movimiento de la vagoneta realizando un RESET al microcontrolador. Sin embargo, ahora el sistema se encuentra protegido frente a una orden incorrecta introducida por el usuario, que podría poner en peligro los instrumentos que se manejan. El nuevo sistema de parada de emergencia no conlleva la realización de un reseteo al sistema, y por consiguiente no implica una posible pérdida de tiempo para volver a inicializarlo. Además, cabe destacar que también cubre un posible fallo de discrepancia de información entre la variable "aux2", que es la que guarda la posición actual, y la posición real en la que se encuentra la vagoneta.

4.5.6 FUNCIÓN “GUARDA_EEPROM”

El diagrama de flujo de la función “GUARDA_EEPROM” se incluye en la Figura 4.27.

Figura 4.27 Diagrama de flujo de la función “GUARDA_EEPROM”

Esta función se encarga de guardar la posición de la vagoneta que se le indica. Puesto que la EEPROM del microcontrolador sólo guarda variables de 1 *byte*, cuando se quiere guardar en la memoria variables de 2 *bytes* ó más *bytes*, se necesitará dividir dichas variables en 2 o más *bytes* según su tamaño.

4.6 FUNCIONES DE ENVIO DE INFORMACION AL LABVIEW

En este apartado se ha agrupado la explicación de las funciones “ENVIO_POSICION_LABVIEW” y ENVIO_FIN_CARRIL_LABVIEW”. Se han incluido los dos flujogramas dentro del mismo apartado ya que sus estructuras son iguales. Se explicará el diagrama de flujo de una y se hará mención a la diferencia que las distingue. Los diagramas de flujo se muestran en las Figuras 4.28 y 4.29. Estas funciones tienen como objetivo informar al programa LabVIEW (desde el microcontrolador) de la posición actual de la vagoneta y del final del extremo derecho del carril. En sus diagramas de flujo aparece explícitamente el símbolo de comunicación entre ambos dispositivos que se creó en el capítulo anterior.

En primer lugar, se comprueba si se está en modo USB o no. Si el microcontrolador y el PC a través del programa LabVIEW no se encuentran conectados, es inútil enviar por el puerto serie cualquier tipo de información. Una vez que se sabe que los dos aparatos están conectados [7], se comprueba si el microcontrolador ha sido enumerado por el programa LabVIEW. A menudo, ocurre que tenemos conectado el cable del USB pero el programa desarrollado no se está ejecutando.

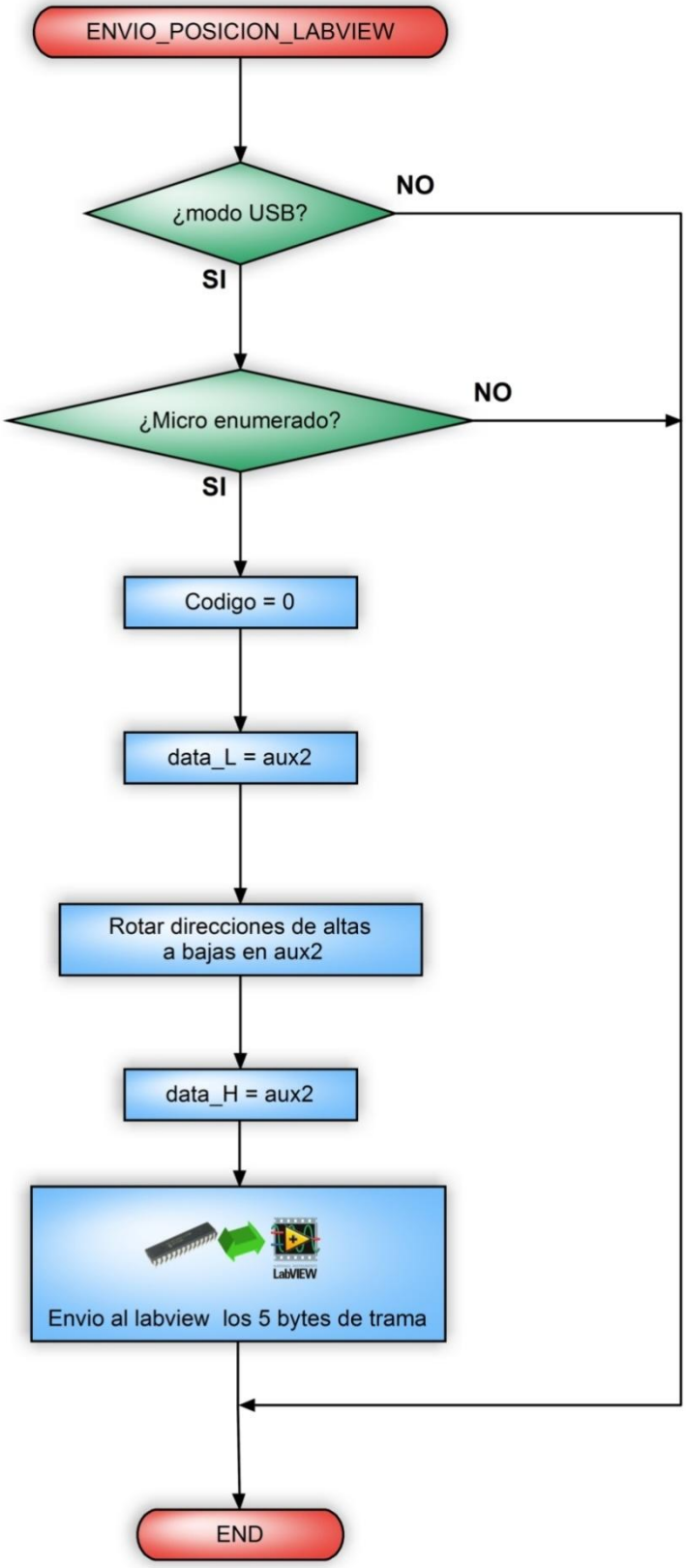


Figura 4.28 Diagrama de flujo de la función "ENVIO_POSICION_LABVIEW"

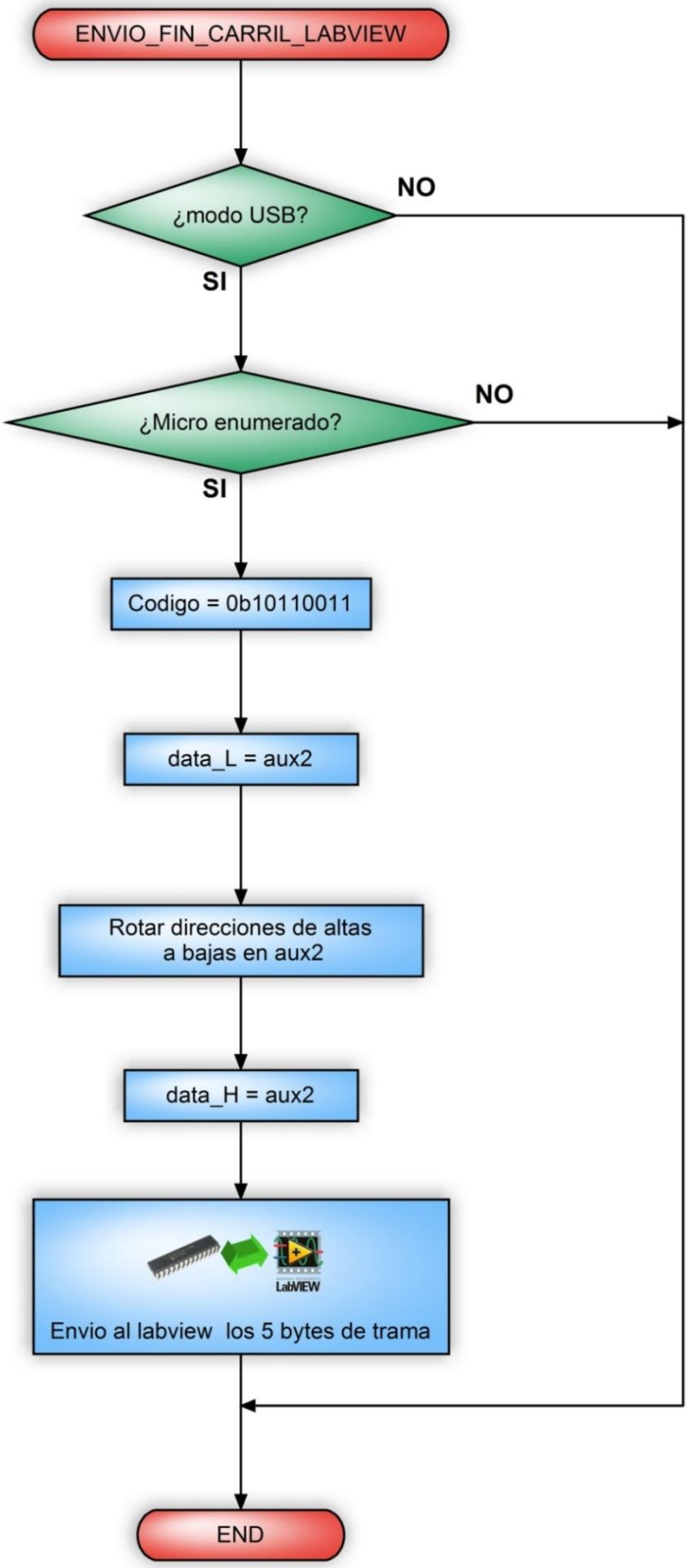


Figura 4.29 Diagrama de flujo de la función "ENVIO_FIN_CARRIL_LABVIEW"

Tras estas dos verificaciones, las variables del protocolo de comunicación (Código, data_H y data_L) toman valores. En la función “ENVIO_POSICION_LABVIEW” la variable “código” toma el valor 0b00000000, como se indicó en el apartado 3.3.2. En el caso de la función “ENVIO_FIN_CARRIL_LABVIEW”, la variable “código” toma el valor en binario 0b10110011. Esta es la única diferencia entre las dos funciones. Por último, se envían los 5 *bytes* correspondientes al protocolo de comunicación.

4.7 FUNCIONES DEL PUERTO USB

En esta sección se han agrupado las funciones relacionadas con la comunicación por el puerto USB. Dichas funciones son:

- ”USB_CONECTADO”
- ”ACCIONES_USB”

4.7.1 FUNCIÓN “USB_CONECTADO”

En este apartado se detallan las acciones que realiza la función “USB_CONECTADO” mediante su flujograma (ver Figura 4.30).

Como ya se comentó en el diagrama de flujo de la función “MAIN”, esta función se llama constantemente, puesto que la comunicación mediante USB es uno de los pilares del presente proyecto. Antes de comentar todo el diagrama, se recuerda que en la función “MAIN” se inicializó la variable “modo”, que es la encargada de llevar la información correspondiente al estado del USB:

- modo = 0 → USB NO CONECTADO
- modo = 1 → USB CONECTADO

Para verificar que el cable USB (PC – MICROCONTROLADOR) ha sido conectado, se debe leer del puerto ADC correspondiente (puesto que el pin de alimentación del cable está conectado a una entrada ADC del microcontrolador). Una vez leído el ADC, se pasa a comprobar si el valor de lectura se corresponde con una tensión similar a la del puerto USB. Si es así, y previamente no se había conectado el cable (modo NO USB), se cambiará el estado de la variable a modo USB, se mostrará en la pantalla LCD el mensaje de la Figura 4.31 y el programa se quedará esperando a que el programa LabVIEW se comunique con él. [6]

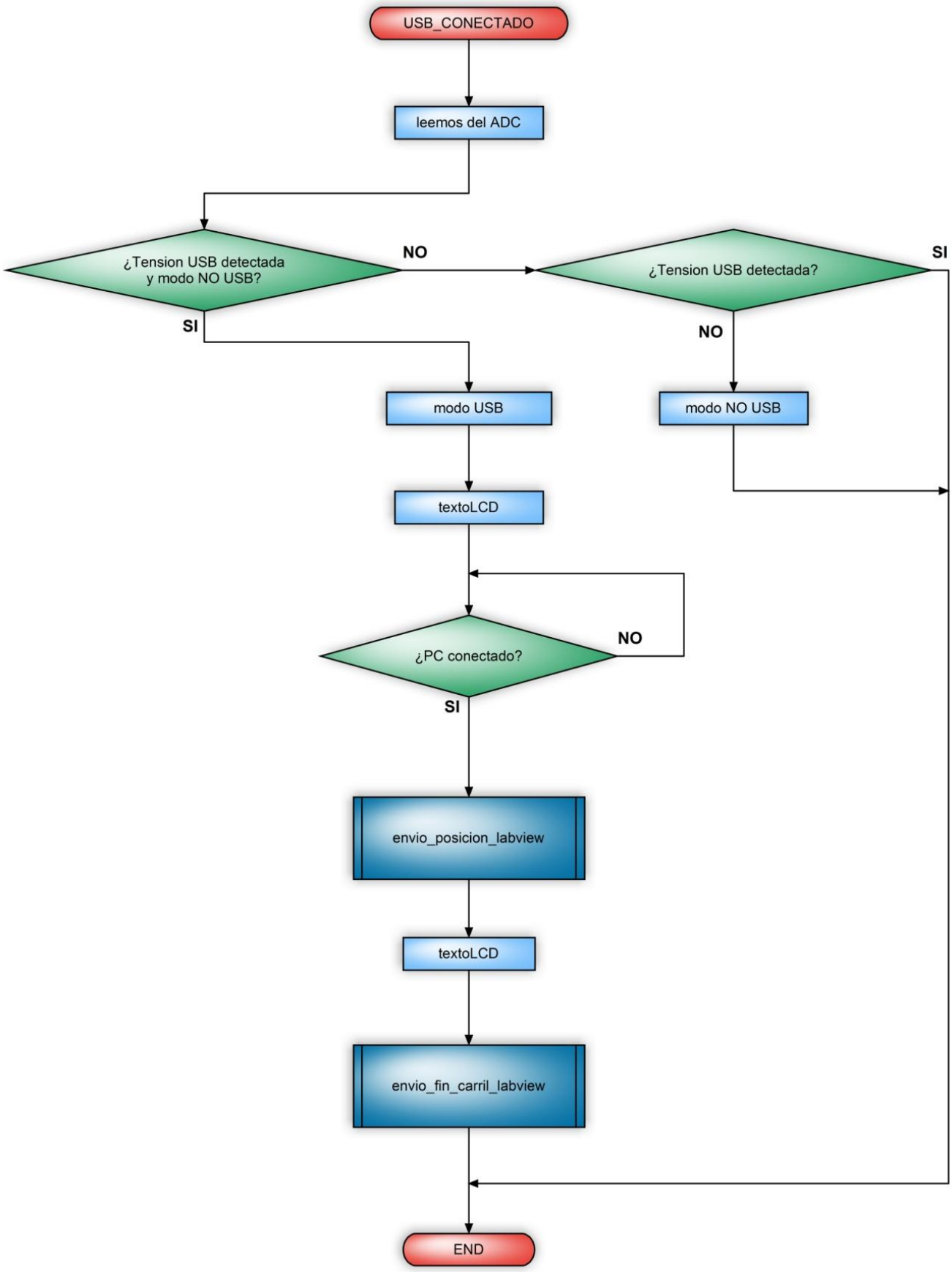


Figura 4.30 Diagrama de flujo de la función "USB_CONECTADO"

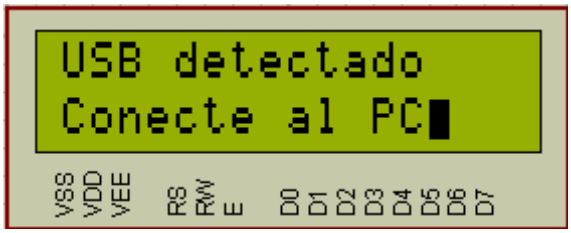


Figura 4.31 Mensaje en la pantalla LCD que muestra el USB detectado

Una vez que el microcontrolador y el programa LabVIEW se reconocen, el microcontrolador llamará a las funciones “ENVIO_POSICION_LABVIEW” y “ENVIO_FINAL_CARRIL_LABVIEW” que son las responsables de enviar toda la información al programa LabVIEW.

Por otro lado, si se desconecta el cable USB mientras se estaba en modo USB, no se cumplirá la primera condición, pasando a comprobar la segunda. En este caso, al estar previamente en modo USB y al no detectarse tensión, se procederá a cambiar de estado (se pasará del estado modo USB al estado modo NO USB).

Por último, si se detecta que hay tensión por el USB y estamos en modo USB, la función rechazará la primera condición, y aceptará la segunda, no ejecutando ninguna acción y por tanto no se realizará ningún cambio.

4.7.2 FUNCIÓN “ACCIONES_USB”

En este último apartado, dedicado a la programación del microcontrolador, se detalla la función “ACCIONES_USB” cuyo flujograma viene representado en la Figura 4.32.

En primer lugar se comprueba si se está en el modo USB. En caso negativo, no se realizará ninguna acción. Por el contrario, si el microcontrolador se encuentra conectado al PC a través del programa LabVIEW, se verificará si en el buffer de entrada existe algún dato [6]. Si no hay dato, la función terminará sin hacer nada. Sin embargo, si se reciben *bytes* del puerto serie, se procederá a guardarlos en variables que posteriormente serán analizadas.

Una vez recibidos todos los *bytes* existentes en el buffer del microcontrolador, se comprobará si el primero y el último coinciden con el valor asignado al protocolo de comunicación. Si no es así, se rechazarán los *bytes* leídos y la función terminará. Si por el contrario, los *bytes* de inicio y fin coinciden, se procederá a ejecutar las acciones indicadas [7] en el protocolo de comunicación.

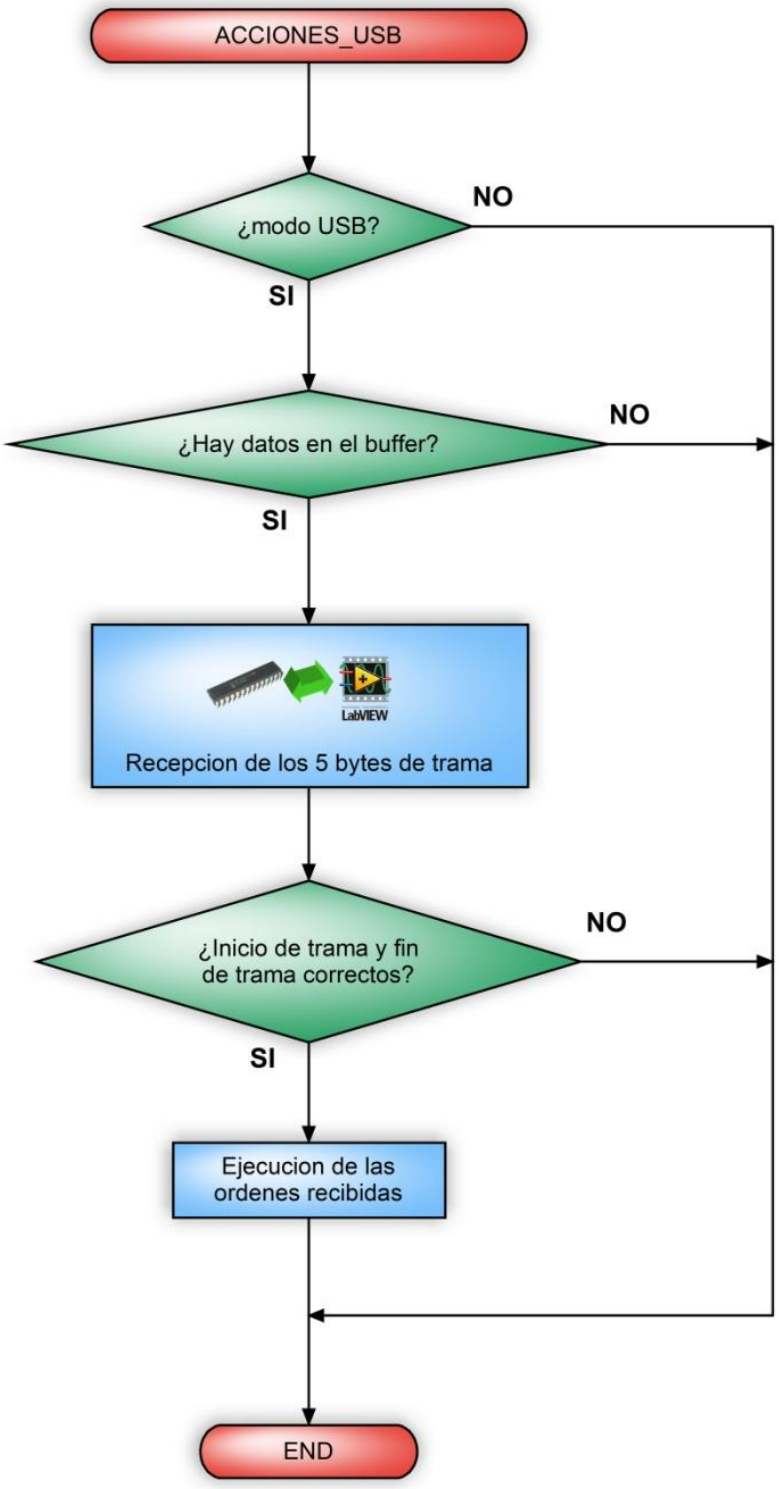


Figura 4.32 Diagrama de flujo de la función "ACCIONES_USB"

Se recuerdan brevemente las acciones a realizar:

Los *bytes* data_H y data_L llevarán la información referente a la posición futura a la que se desea que se desplace la vagoneta, por tanto se juntarán en una variable de 2 *bytes*, y dicha variable se igualará a la variable interna “pos” descrita anteriormente.

El *byte* código, llevará inmerso el sentido de movimiento de la vagoneta, y la velocidad que debe llevar. Tras decodificar este *byte*, los 4 bits referentes a la velocidad indicarán a la función “PASO_MOTOR” el tiempo entre pasos. Esta variable de velocidad quedará guardada, y no se modificará a no ser que desde el programa LabVIEW se indique lo contrario, o se realice un reseteo al microcontrolador, en cuyo caso volverá al valor por defecto. La relación de velocidades según el tiempo de espera entre pasos se detallará en el Capítulo 5.

Por otro lado, se tienen los 4 bits referentes al sentido de movimiento de la vagoneta. En este caso, aunque el programa LabVIEW le está indicando el sentido que llevará la vagoneta, el microcontrolador realiza de nuevo la comprobación. Una vez verificado y teniendo la posición a la que ir en “pos”, se le hará pensar que se ha accionado el pulsador 5 (que ejecuta la acción de movimiento) de la función “LEER_PULSADORES”.

Puede observarse que en la función “MAIN” se llama primeramente a “ACCIONES_USB” antes que a “ACCIONES_MANUALES”. En un primer momento, se podría pensar en llamar a “LEER_PULSADORES” y posteriormente a “ACCIONES_MANUALES” para ejecutar sus debidas acciones, pero de esta manera se evita escribir nuevamente las acciones del pulsador 5 de “ACCIONES_MANUALES” tras la llamada a “ACCIONES_USB”

CAPÍTULO 5

SISTEMA MICROPOSICIONADOR CONTROLADO MEJORADO Y PRUEBAS DE FUNCIONAMIENTO

5.1 MEJORAS INTRODUCIDAS EN EL SISTEMA MECANICO

En el capítulo 5 se describe el sistema microposicionador controlado final y las pruebas de funcionamiento que dan validez al desarrollo de la aplicación en entorno gráfico. En concreto, en este primer apartado, se describen las mejoras introducidas en el sistema mecánico y los motivos que las han generado.

La primera mejora que se abordó fue la sujeción adecuada de los finales de carrera. En el diseño previo a este trabajo, los finales de carrera funcionaban como detectores del alcance del final del carril por la vagoneta. Mecánicamente estaban sujetos sólo con un vástago, lo que podía provocar una rotación en alguno de ellos y provocar un fallo en la medida de la posición absoluta de la vagoneta. En la Figura 5.1 se muestra el sistema microposicionador completo antes de realizar ningún cambio mecánico en este proyecto. En la Figura 5.2 se detalla el modo en que se hallaban sujetos los finales de carrera del diseño previo, para el extremo izquierdo del carril y para el extremo derecho. Se observa que cada vástago se encontraba atornillado a la plancha de aluminio (base de sujeción del sistema motorizado).

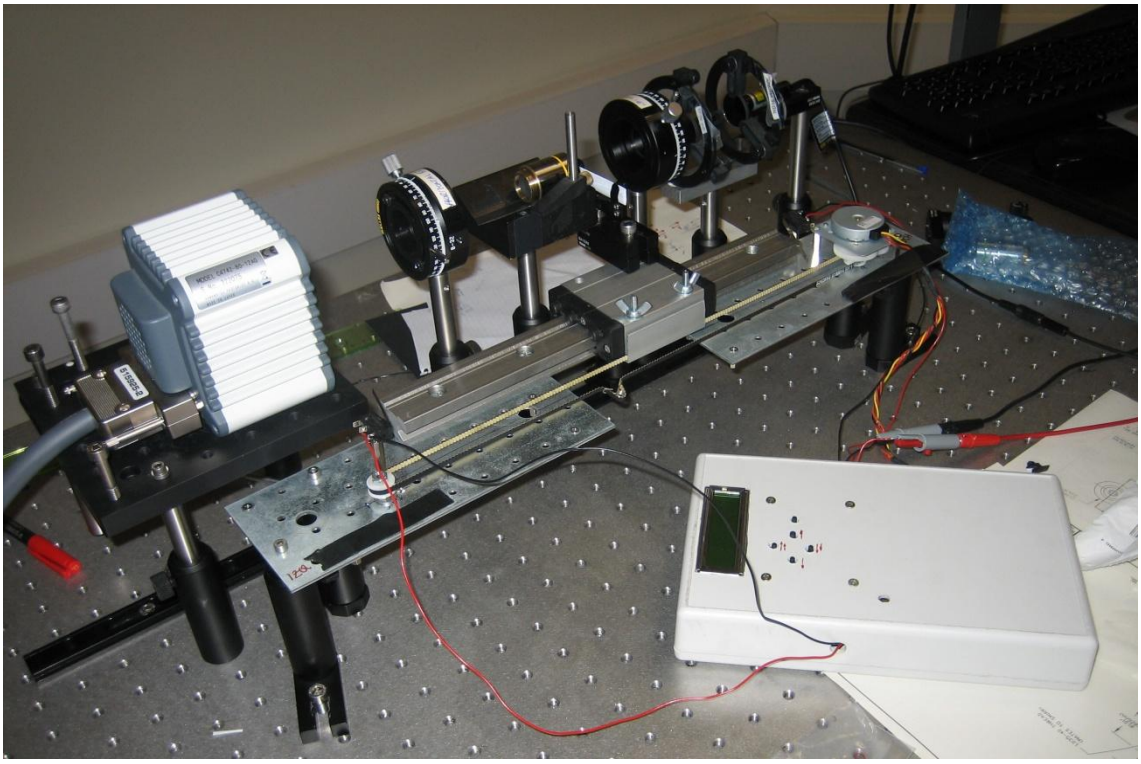
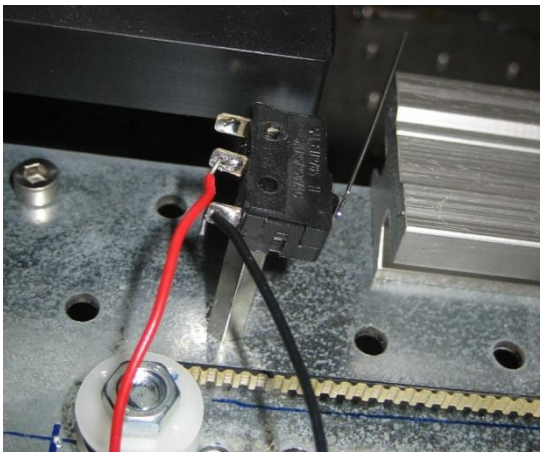
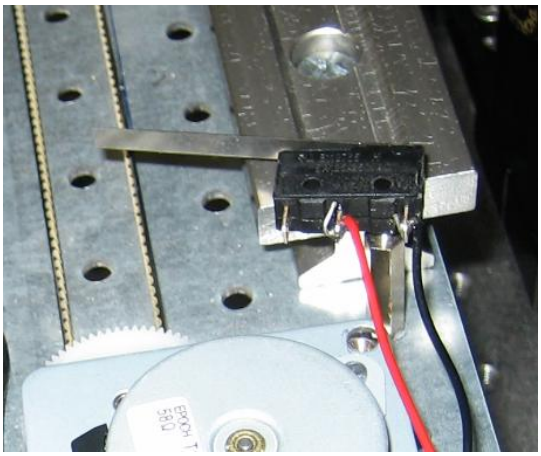


Figura 5.1 Sistema mecánico del microposicionador previo a este proyecto



a) Extremo izquierdo del carril



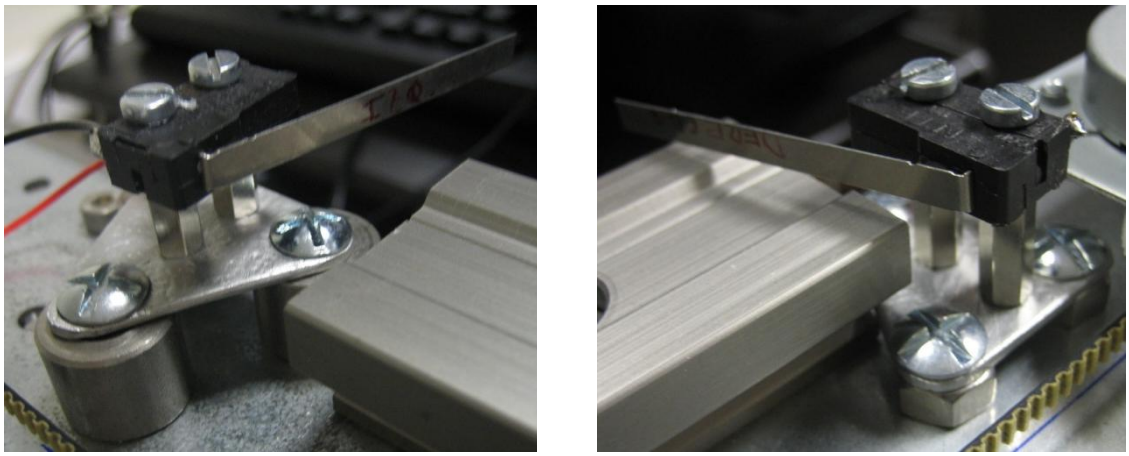
b) Extremo derecho del carril

Figura 5.2 Sujeción del final de carril en el diseño previo a este proyecto

Para la implementación del nuevo sistema mecánico, se consideró adecuado conservar los sensores de final de carrera y optimizar la sujeción de los mismos. Para ello se utilizó una pieza de aluminio adicional que sirvió de soporte nuevo entre el sensor y la base del sistema mecánico (mediante los vástagos). Se escogió una pieza de aluminio triangular agujerada en tres puntos separados con la misma métrica que la base de aluminio (ver Figura 5.3). Además se taladraron dos orificios nuevos en la pieza adaptados para su anclaje al sensor.

Figura 5.3 Pieza de sujeción de los finales de carrera

Una vez montado el final de carril sobre la nueva pieza, y esta a su vez sobre la plancha base del sistema mecánico, los finales de carril quedaron bien sujetos evitando futuras medidas incorrectas. El montaje completo de los finales de carril en los extremos izquierdo y derecho, se muestra en la Figura 5.4. En ella se observa que el anclaje es mucho más estable y válido para conseguir la repetitividad de las medidas.



a) Extremo izquierdo del carril **b) Extremo derecho del carril**

Figura 5.4 Nueva sujeción de los sensores de final de carril

Otra mejora que se ha llevado a cabo en la parte mecánica del sistema, es la implantación de un botón de RESET superficial en la caja donde está incluido el sistema *hardware*. En el diseño previo a este proyecto, sólo se podía acceder al botón de RESET (interior de la caja) insertando un objeto (de diámetro no superior a 5mm) a través de un agujero destinado a tal fin. Al no existir ningún sistema de parada de emergencia, se hizo obligatoria la necesidad de habilitar un acceso más sencillo e inmediato al botón de RESET. La inclusión de un botón de RESET accesible superficialmente permitió el paro automático de la vagoneta estando en movimiento. Adicionalmente, y como se ha explicado previamente, además del botón de RESET se incluyó otro método de parada de emergencia por programa.

En la Figura 5.5 se muestra el nuevo botón de RESET con su etiqueta identificativa.

Figura 5.5 Botón nuevo de RESET accesible desde la superficie de la caja del controlador

Hay que notar que las etiquetas que se ven en las Figuras 5.5 y 5.6, junto a los botones, han sido incluidas para facilitar el significado de cada uno de los pulsadores. Estas pegatinas anteriormente no se incluían; el detalle del panel de la caja del controlador se muestra en la Figura 5.7.

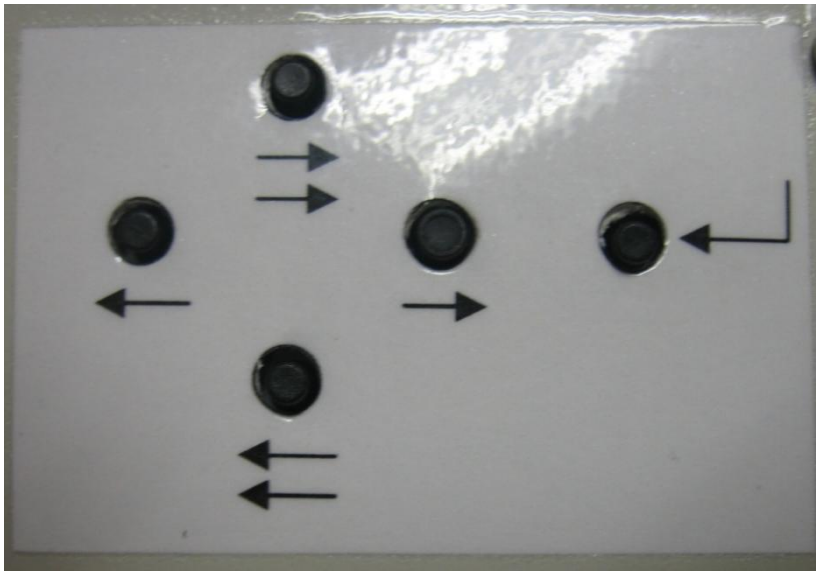


Figura 5.6 Pegatina indicativa del significado de cada botón de acción

Figura 5.7 Sistema microposicionador previo a la inserción del botón de RESET y las pegatinas indicativas

Finalmente, se ha mejorado también la conexión entre la caja de control y la parte mecánica del sistema. Para ello, se han insertado unas clemas que facilitan la desconexión rápida entre la caja controladora y los finales de carril (ver Figuras 5.8 y 5.9).

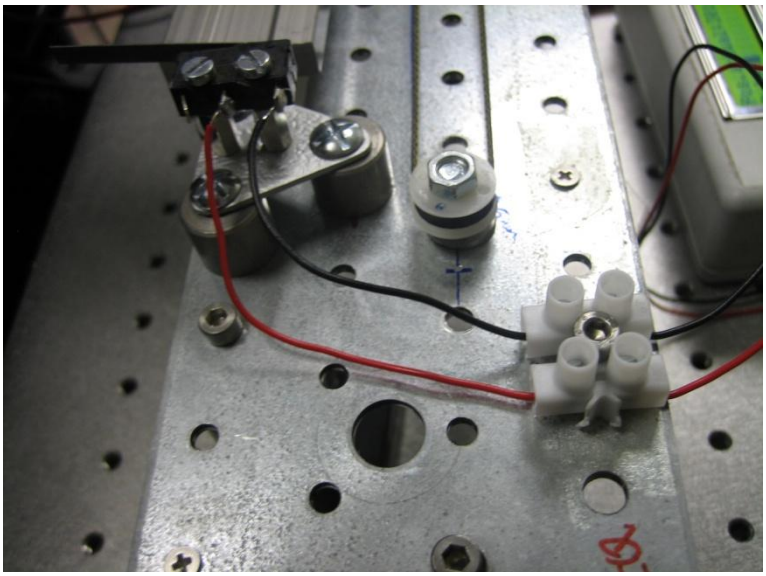


Figura 5.8 Nueva regleta del sensor final de carril izquierdo

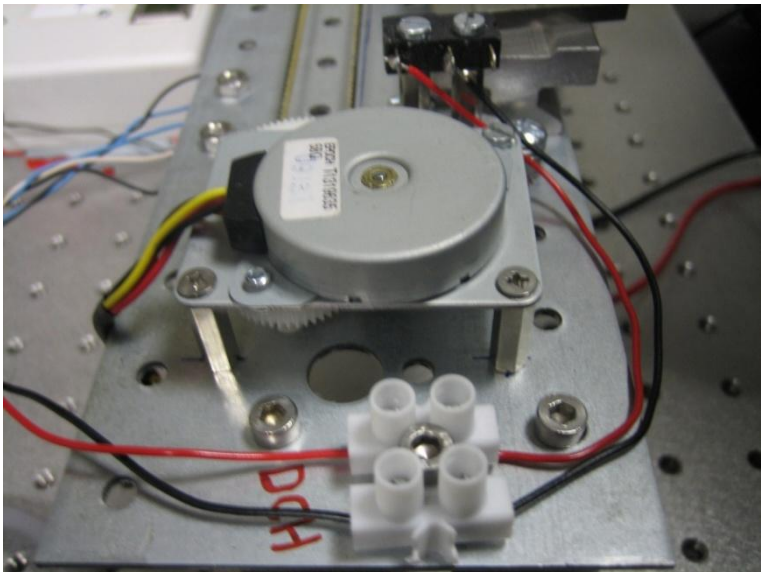


Figura 5.9 Nueva regleta del sensor final de carril derecho

Asimismo, se ha incluido otra clema a modo de regleta (ver Figura 5.10), para conectar y desconectar fácilmente el controlador del motor paso a paso. Con esta mejora se hace más cómodo el manejo de la parte mecánica y del controlador y, llegado el caso, su traslado de forma independiente. En el sistema previo, si el controlador quería ser trasladado individualmente, debía ser abierto para desconectar los terminales.

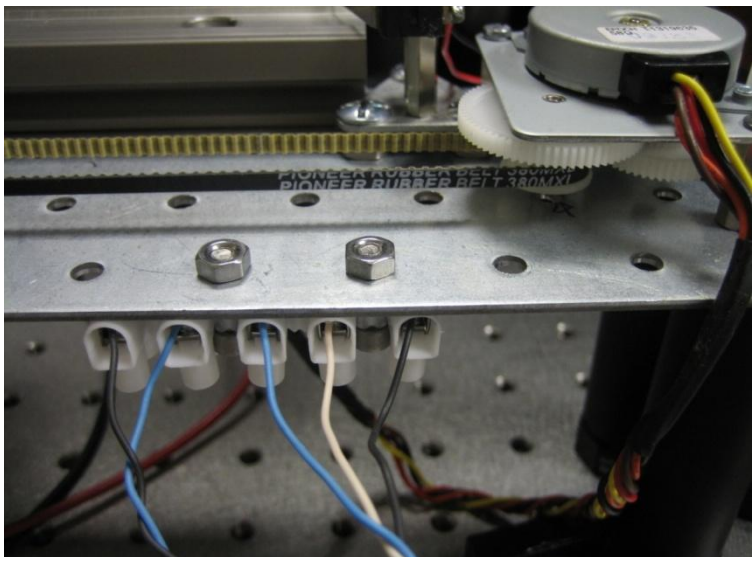


Figura 5.10 Nueva clema a modo de regleta para conectar el controlador y el motor paso a paso

Tras haber realizado las mejoras descritas, el sistema mecánico-*hardware* electrónico resultante presenta el aspecto que se representa en la Figura 5.11.

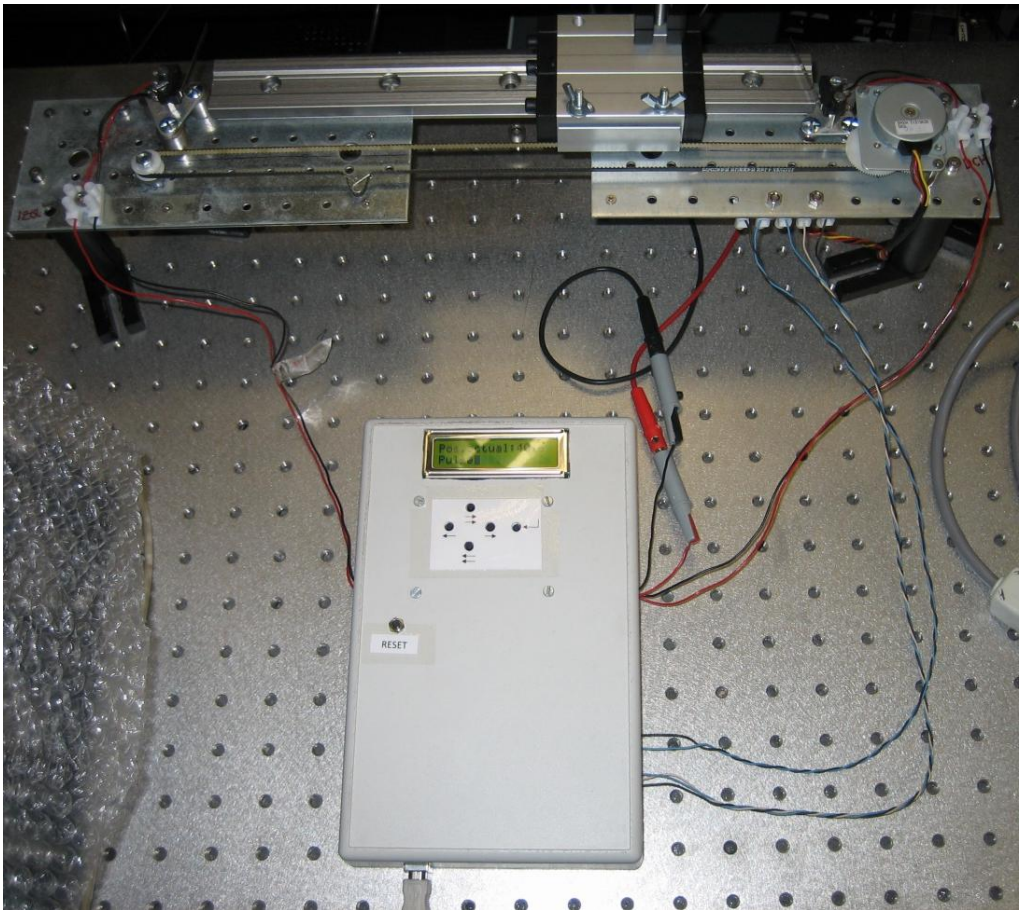


Figura 5.11 Sistema mecánico-*hardware* electrónico completo

5.2 GUIA DE USUARIO DE LA APLICACIÓN

La inclusión de este apartado persigue un objetivo doble. Por una parte, ilustrar de forma visual todas las funcionalidades de la aplicación en entorno gráfico desarrollada para uso de los futuros desarrolladores de trabajos futuros. Por otra, guiar a un usuario que utilice la aplicación en el uso de la misma. Para llevar a cabo este doble objetivo se ha considerado adecuado plantear esta sección a modo de guía de usuario de la interfaz gráfica desarrollada. Se explicarán de forma secuencial los pasos a seguir para el arranque de la aplicación, para conectar y desconectar el microcontrolador al PC, los parámetros que se visualizan y los campos en los que se introduce información para el movimiento del motor. También se detallarán los pasos que el usuario deberá realizar en caso de una desconexión involuntaria o una mala conexión por parte del LabVIEW o del microcontrolador.

5.2.1 ARRANQUE Y CIERRE DE LA APLICACION

En este apartado se detallan los pasos que debe seguir el usuario final para hacer funcionar el sistema completo (LABVIEW-MICROCONTROLADOR) de una manera rápida, sencilla y efectiva.

En primer lugar, el ordenador (PC) con el que se conectará el microposicionador debe tener instalado el programa LabVIEW y el fichero de la aplicación desarrollada. También es preciso disponer de un cable USB del tipo A-B (ver Figura 5.12), con el que se hará la futura conexión.

Figura 5.12 Cable USB tipo A-B

Con el *software* disponible y el cable USB se puede proceder a conectar ambas partes de acuerdo al orden que se describe a continuación. Se trata de 5 pasos que deben realizarse de forma secuencial.

PASO 1

Conectar el cable USB entre el PC y el microcontrolador (el conector USB por el lado del microcontrolador se encuentra accesible en la caja del controlador). Una vez que se ha detectado la tensión del USB, en la pantalla LCD del microcontrolador aparecerá el mensaje “USB DETECTADO, CONECTE AL PC” (ver Figura 5.13). Esto indica que el microcontrolador se queda en espera hasta que el programa desarrollado se ejecute.

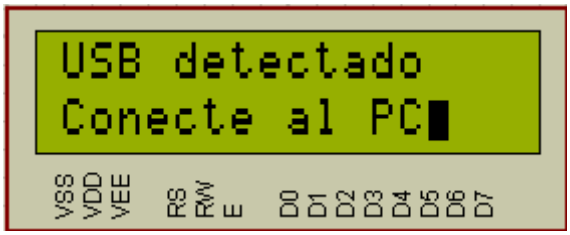


Figura 5.13 Conexión entre PC y microcontrolador

PASO 2

Abrir el proyecto de la aplicación desarrollada en LabVIEW (ver Figura 5.14).

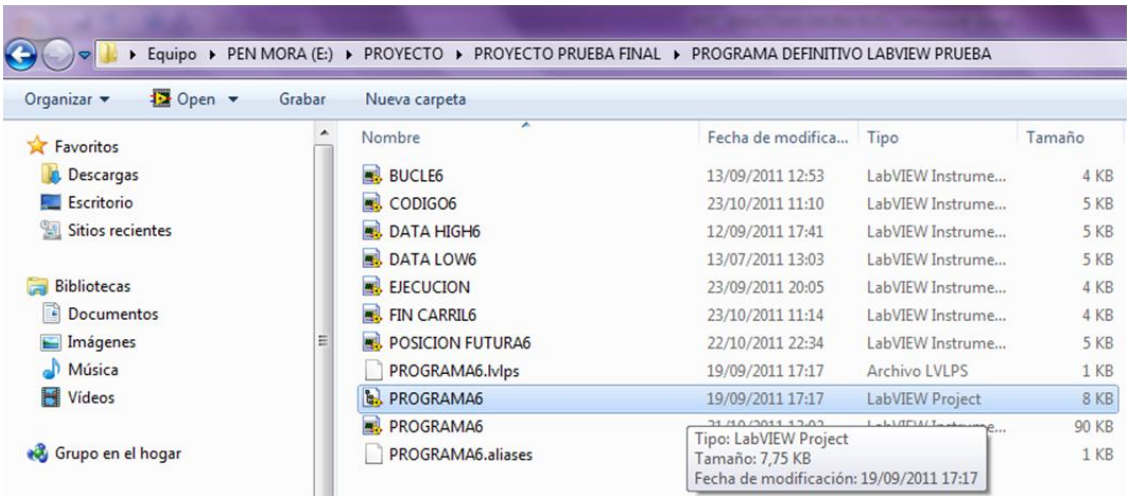


Figura 5.14 Proyecto de la aplicación desarrollada en LabVIEW

A continuación, abrir el programa propio de la aplicación de control del sistema microposicionador (ver Figura 5.15). Dentro del proyecto se pueden incluir diferentes ficheros de trabajo y elegir el que se desea ejecutar.

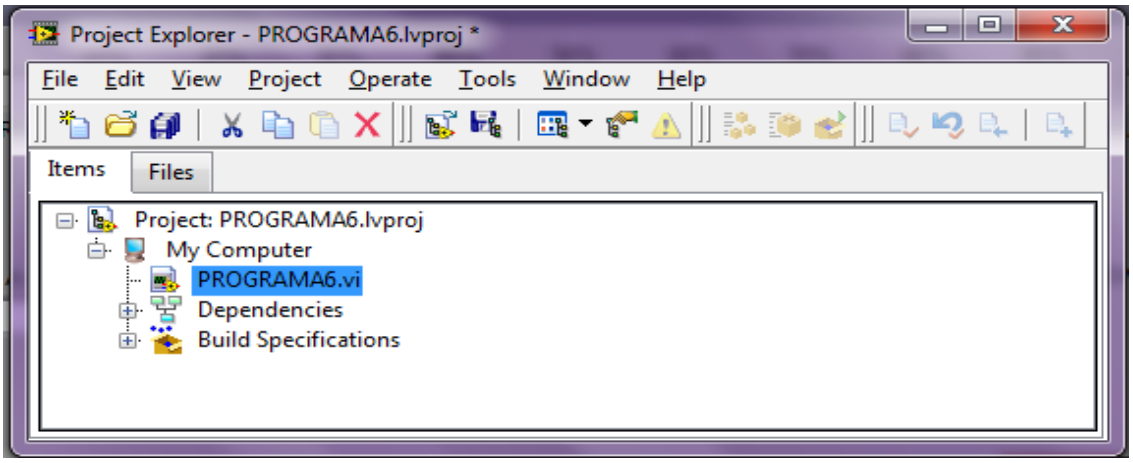


Figura 5.15 Programa de la aplicación desarrollada en LabVIEW

También se puede abrir el programa (Instrumento Virtual) directamente sin necesidad de abrir en primer lugar el proyecto, si no cliqueando directamente sobre el programa como se indica en la Figura 5.16.

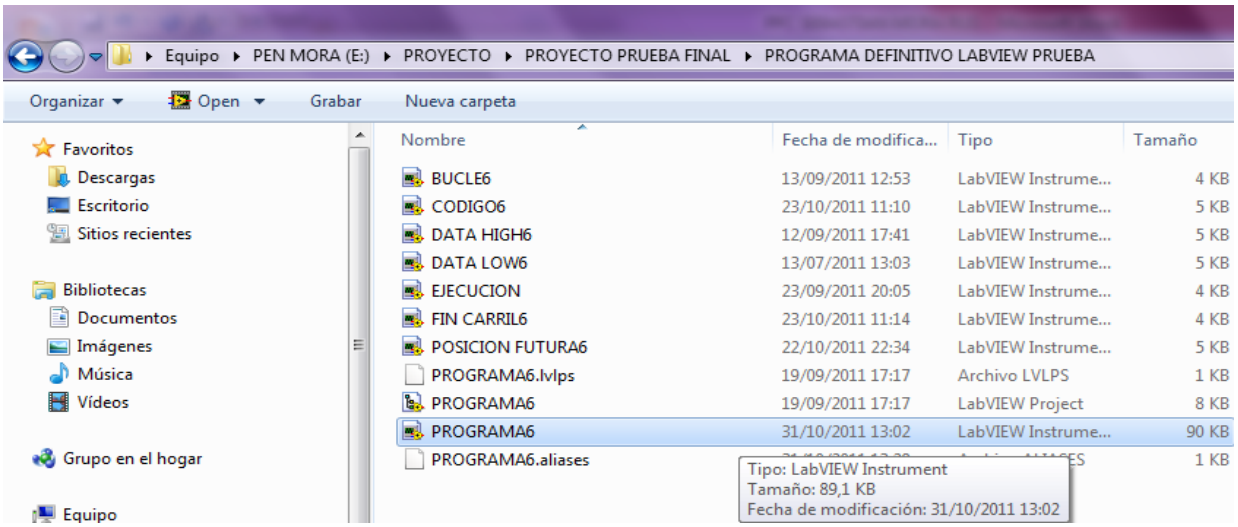


Figura 5.16 Acceso directo al instrumento Virtual (VI) desarrollado

PASO 3

Una vez abierto el Instrumento Virtual programado se muestra el panel frontal del mismo. Este panel frontal consta de 4 pestañas: “PRINCIPAL”, “INICIALIZACIÓN”, “PARÁMETROS USB” y “VISUALIZACIÓN”. Por defecto al abrir el programa aparece accesible la pestaña “PARAMETROS USB” tal como se ve en la Figura 5.17. Dentro de dicha pestaña se selecciona el puerto COM al que ha sido conectado el microposicionador desplegando el menú “PUERTO COM MICROCONTROLADOR”.

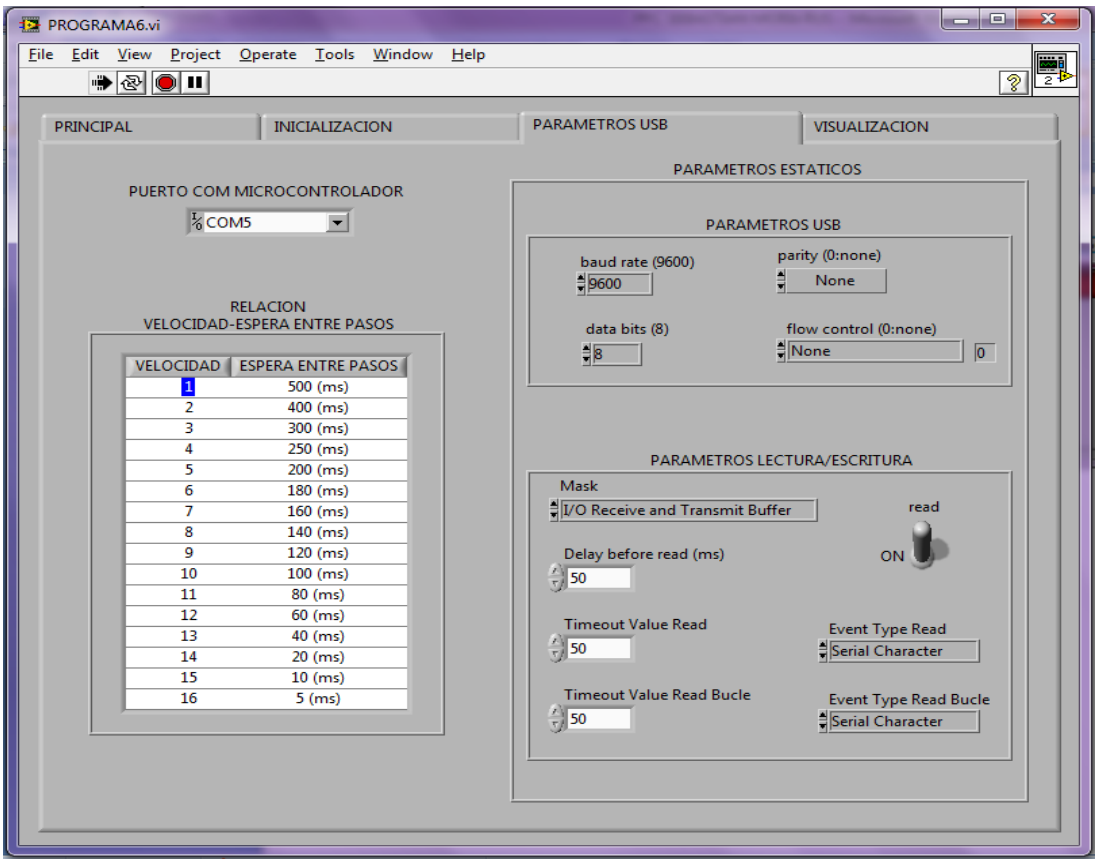


Figura 5.17 Panel frontal inicial al abrir la aplicación (Pestaña USB activa)

En el menú desplegable (ver Figura 5.18.) aparecen varios puertos COM para elegir uno y la opción "*Refresh*", para actualizar los posibles puertos disponibles.

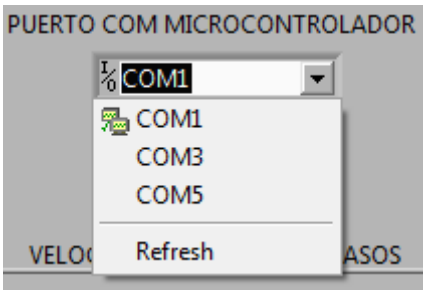


Figura 5.18 Selección del puerto COM

PASO 4

Una vez que se ha seleccionado el puerto COM correspondiente, se elegirá la pestaña "PRINCIPAL" que constituye la interfaz entre el usuario y el microcontrolador. Con la pestaña "PRINCIPAL" activa, se ejecuta la aplicación pinchando en "Operate → Run" (ver Figura 5.19) o cliqueando en el botón con forma de flecha mostrado en la Figura 5.20.

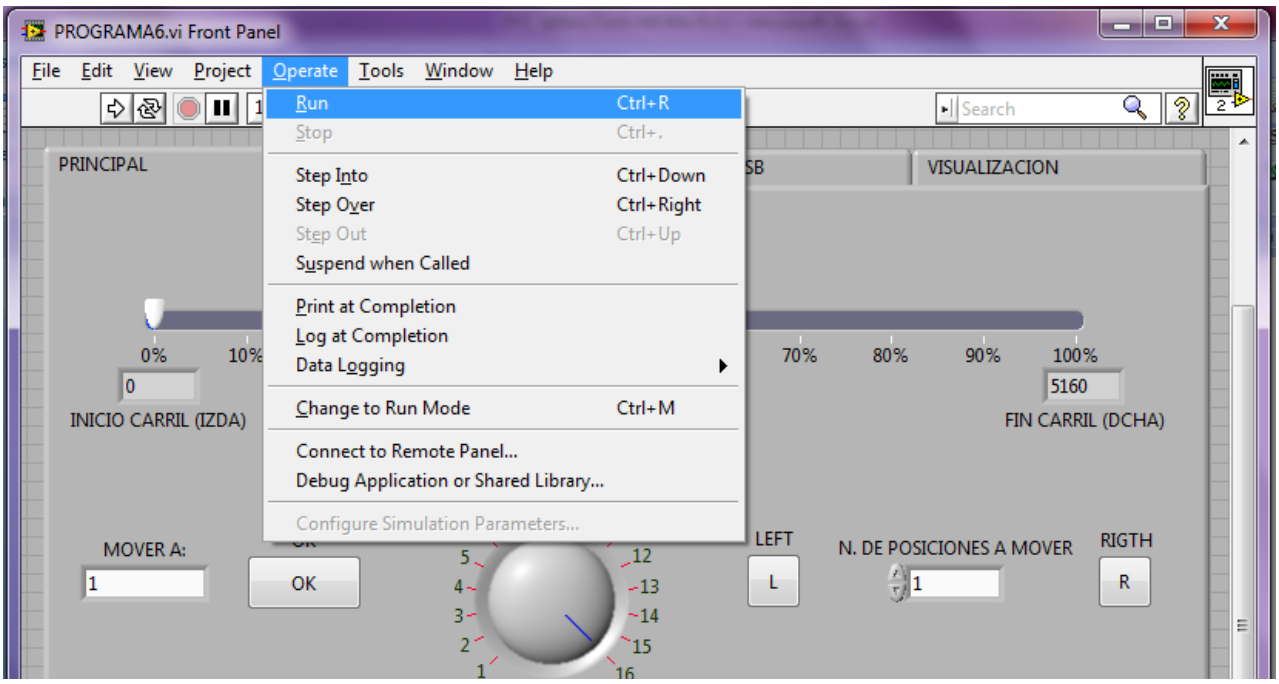


Figura 5.19 Ejecutar el LabVIEW mediante Operate → Run

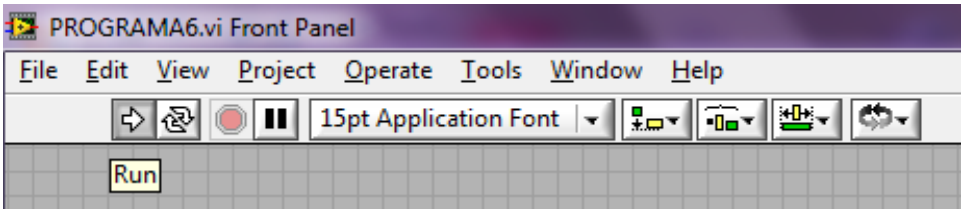


Figura 5.20 Ejecutar el LabVIEW mediante el icono de "Run"

Tras ejecutar el Paso 4, desaparece el mensaje mostrado anteriormente en la pantalla LCD (Figura 5.13) y aparece un mensaje con la posición actual en la que se encuentra la vagoneta. En la Figura 5.21 se muestra un ejemplo del mensaje mostrado en la pantalla LCD de la posición 80 ocupada por la vagoneta.

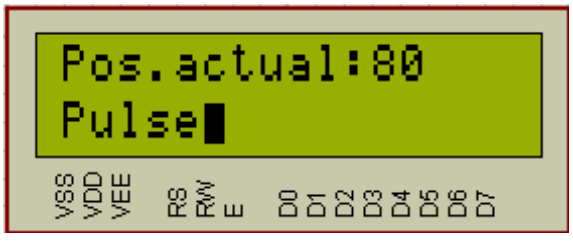


Figura 5.21 Mostrar posición actual

PASO 5

Cuando el usuario ha acabado de utilizar el *software* de control, es muy importante que cierre la aplicación de manera correcta. Es decir, una vez que ha finalizado su trabajo, debe parar la aplicación pulsando el botón de “STOP” (detallado en apartado 5.2.2) que viene coloreado de rojo en la pestaña “PRINCIPAL” de la aplicación. Si por algún casual no se cierra la aplicación adecuadamente, se puede consultar los pasos a seguir en el apartado 5.3.2.

5.2.2 DEFINICION DE LOS CAMPOS DE MOVIMIENTO

En esta parte de la guía se definirán todos y cada uno de los parámetros (Campos) de los que consta la aplicación desarrollada. La interfaz de usuario consta de 4 pestañas bien diferenciadas: “PRINCIPAL”, “INICIALIZACION”, “PARAMETROS USB” y “VISUALIZACION” tal como se representa en la Figura 5.22.

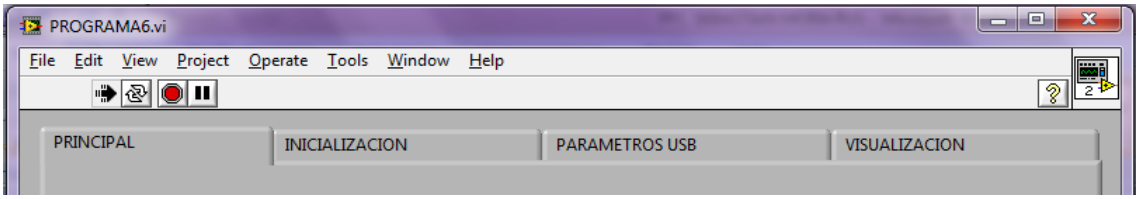


Figura 5.22 Pestañas de la aplicación

Pestaña “PRINCIPAL”

La que más utilizará el usuario final, será la pestaña llamada “PRINCIPAL” (ver Figura 5.23). En ella están incluidos todos los botones y visualizadores necesarios para el control del microposicionador. Además aparecen diferentes campos que el usuario debe completar para definir la secuencia de movimiento del motor. Para distinguir estos campos a completar, se muestran con fondo de color blanco aquellos en los que es posible la introducción de información. A continuación se describe cada uno de ellos.

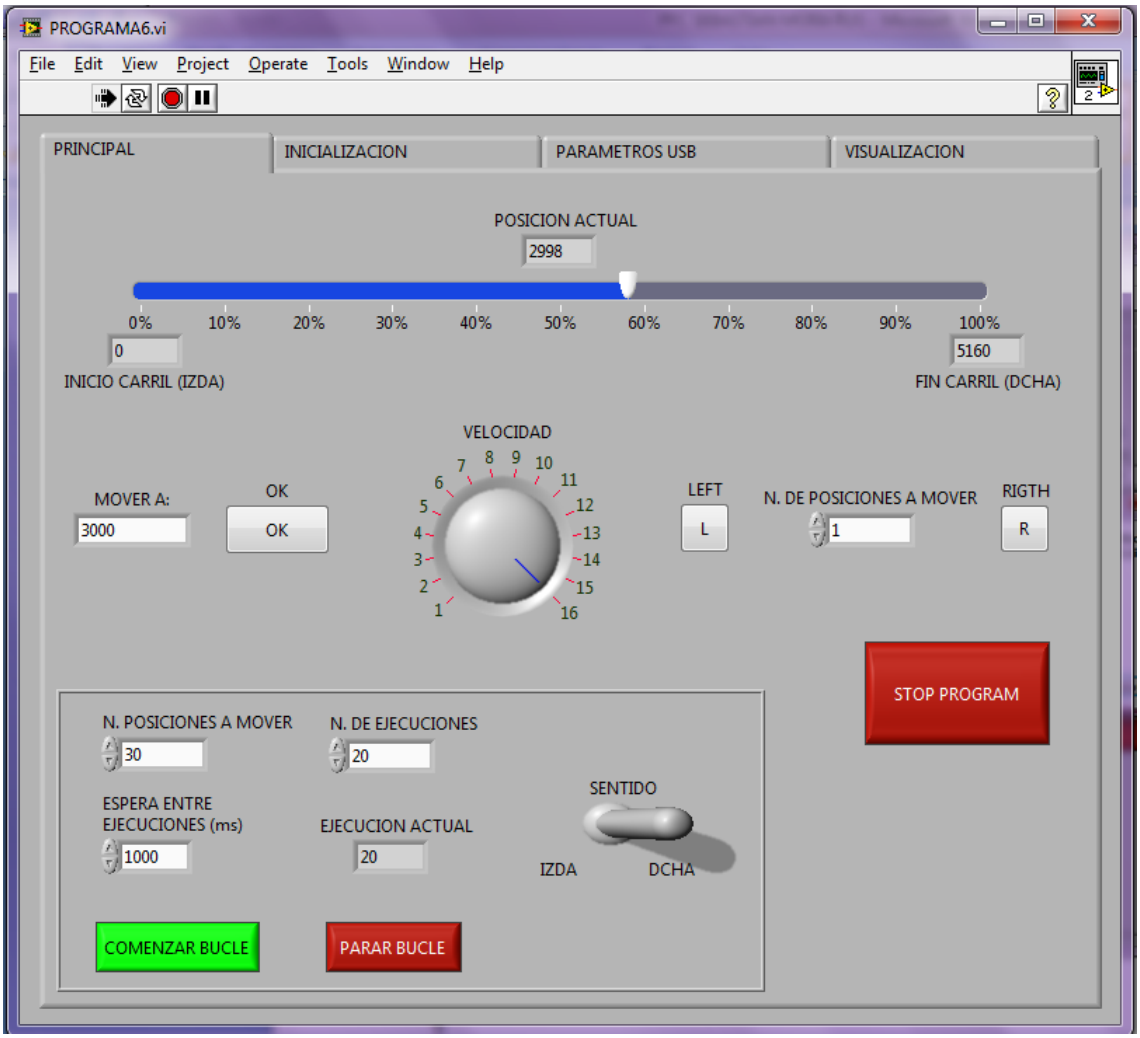


Figura 5.23 Pestaña principal de la interfaz desarrollada

En primer lugar, empezando a describir la ventana desde la parte superior a la zona inferior, aparece una barra asociada a tres visualizadores (ver Figura 5.24). Estos visualizadores han sido creados para mostrar una información similar a la que se presenta en la pantalla LCD sobre las propiedades del carril y la posición de la vagoneta.

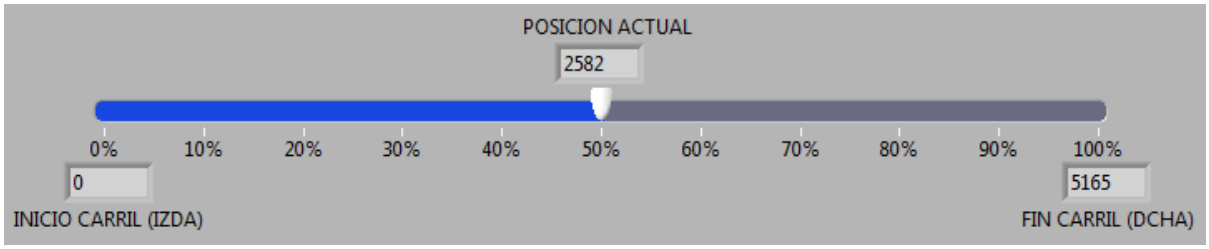


Figura 5.24 Visualizador de posición actual de la vagoneta y extremos del carril

El visualizador de la izquierda muestra el valor del extremo izquierdo del carril (“INICIO CARRIL”), que siempre será 0. El central (“POSICIÓN ACTUAL”), muestra la posición actual de la vagoneta en todo momento y siempre referenciado respecto al extremo izquierdo (INICIO CARRIL). El de la derecha (“FIN CARRIL”), dará la información correspondiente al extremo derecho del carril. La barra representa, en tanto por ciento, la posición en la que se encuentra la vagoneta. Este porcentaje viene determinado en función del ancho del carril, puesto que éste puede ser cambiado. Es decir, el usuario final puede trabajar sólo en un rango determinado dentro del total, acotándolo en la inicialización. En la Figura 5.25 se ilustra un ejemplo de disminución en el rango de movimiento de la vagoneta sobre el carril respecto al caso anterior. En la Figura 5.24, “FIN CARRIL” vale 5165, mientras que en el caso del ejemplo, Figura 5.25, “FIN CARRIL” vale 103.

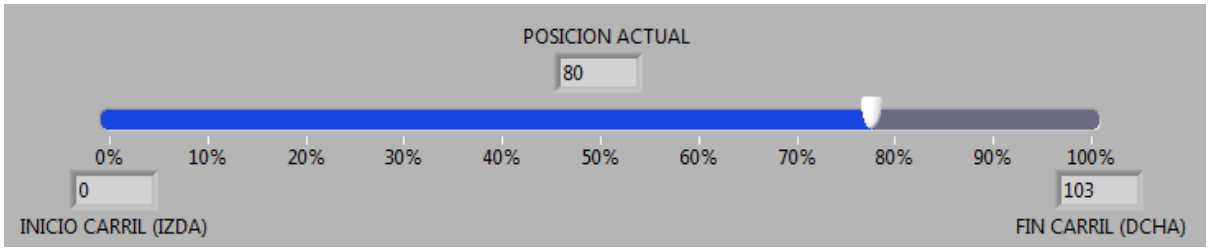


Figura 5.25 Disminución del rango de trabajo de la vagoneta

Debajo de la barra descrita, se han incorporado una serie de botones encargados de enviar órdenes al microcontrolador. Estos se han diseñado siguiendo las funciones de los pulsadores implementados físicamente en la caja del controlador, para que su utilización sea lo más intuitiva posible (ver Figura 5.26).

Figura 5.26 Botones de envió de órdenes al microcontrolador

Los dos primeros iconos, a la izquierda de la Figura 5.26, son los encargados de realizar movimientos largos de la vagoneta. En el campo “MOVER A:” se introduce el valor deseado de la posición futura de la vagoneta y con el botón “OK” se envía dicha información al microcontrolador (ver Figura 5.27).

Figura 5.27 Campo "MOVER A:" y pulsador OK para movimientos largos de la vagoneta

El selector circular central indica la "VELOCIDAD" que llevará la vagoneta. Esta información es enviada cada vez que se desea mover la misma. El selector colocado en la posición número 1 define la velocidad más lenta en el movimiento del motor y en el número 16, la velocidad máxima de desplazamiento por el carril. Cada vez que se ejecuta la aplicación, el selector circular se posicionará, por defecto, en el valor 16 tal como se aprecia en la Figura 5.28.

Figura 5.28 Selector circular central de velocidad de movimiento de la vagoneta

Por último, están los iconos representados en la Figura 5.29. Aquí se observan 2 botones y un campo visualizador para introducir números. Los botones sirven para indicar en qué sentido se moverá la vagoneta, y en el campo central se introducirá el numero de pasos de movimiento. Estos iconos realizan la misma función que los pulsadores de movimientos cortos de la caja del sistema controlador. Estos iconos tienen una ventaja añadida, puesto que en el microposicionador el valor constante de los recorridos cortos es 1, y en la aplicación, este valor puede ser configurado por el usuario final.

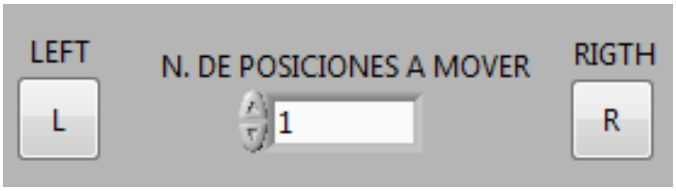


Figura 5.29 Campo "N. DE POSICIONES A MOVER" y pulsadores L y R para movimientos cortos de la vagoneta

Debajo de los botones asociados a movimientos cortos del motor, se encuentra uno de los botones clave del *software* de control desarrollado. Este botón es el encargado de parar la ejecución del programa de una manera ordenada (ver Figura 5.30).

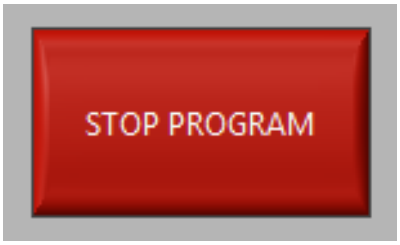


Figura 5.30 Botón de STOP PROGRAM

Gracias a este botón, se cierra el puerto serie de forma correcta y se evitan futuras conexiones fallidas entre el microcontrolador y el PC mediante el programa LabVIEW. El programa se cierra incorrectamente en los dos siguientes casos:

- Abortar la ejecución (ver Figura 3.9).
- Ó cerrar la aplicación directamente.

Los últimos botones y visualizadores que alberga la pestaña “PRINCIPAL” se sitúan en la parte inferior de la ventana dentro de un recuadro para poder diferenciarlos del resto. Estos son los parámetros de control del bucle automático programado (ver Figura 5.31).

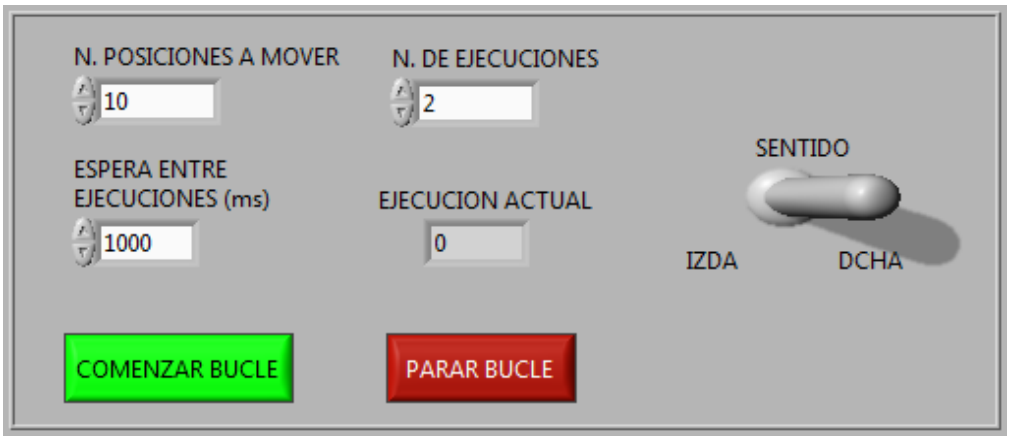


Figura 5.31 Botones y visualizadores de control de ejecución del bucle

El campo “N. POSICIONES A MOVER” representado en la Figura 5.32 determina el número de pasos que la vagoneta se va a mover para una sola ejecución del bucle.

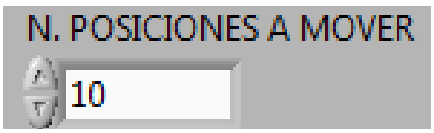


Figura 5.32 Numero de posiciones a mover por la vagoneta en una ejecución del bucle

El parámetro configurable “N. DE EJECUCIONES”, representado en la Figura 5.33, indica el número de veces que se desea ejecutar el bucle.

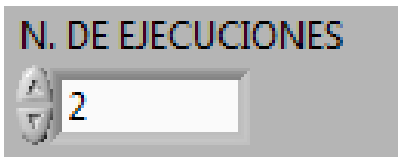


Figura 5.33 Numero de ejecuciones del bucle

Otro campo configurable es el representado en la Figura 5.34. En dicho campo se introduce el tiempo (en milisegundos) que la vagoneta estará parada entre cada ejecución del bucle. Es decir, es el tiempo que espera parada la vagoneta desde que llega a una posición determinada hasta que se vuelve a mover hacia otra posición diferente.

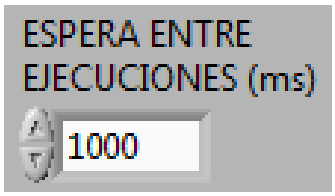


Figura 5.34 Tiempo de espera de la vagoneta entre movimientos

El visualizador “EJECUCION ACTUAL” muestra el número ordinal de la ejecución del bucle. Este no es configurable ya que sólo indica al usuario la ejecución que se está llevando a cabo (ver Figura 5.35).

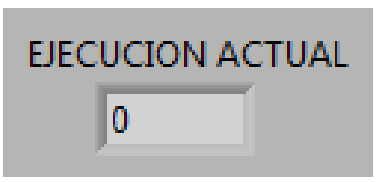


Figura 5.35 Indicador del número ordinal de ejecución del bucle

El último de los parámetros configurables para la ejecución del bucle, viene representado en la Figura 5.36. Con él, el usuario indica el sentido de movimiento de la vagoneta en que se va a realizar el bucle repetitivo.

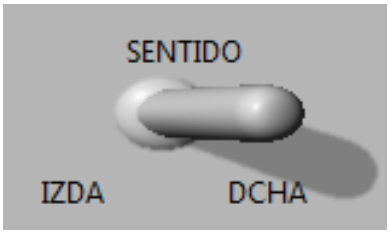


Figura 5.36 Sentido de movimiento de la vagoneta en la ejecución del bucle

Una vez que el usuario ha configurado todos los parámetros de control, existen dos botones que ordenan la ejecución y la parada del bucle. El primero puede verse en la Figura 5.37; y está destinado a iniciar la ejecución del bucle.

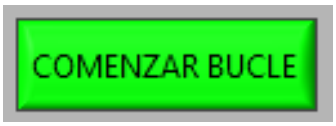


Figura 5.37 Botón de comienzo de ejecución del bucle

El segundo tiene como función parar la ejecución del bucle en caso de haber iniciado el bucle con unos parámetros incorrectos (ver Figura 5.38), por ejemplo.

Figura 5.38 Botón de parada de ejecución del bucle

Pestaña “INICIALIZACIÓN”

En esta nueva pestaña llamada “INICIALIZACION”, se incluye solamente un botón que se llama igual (ver Figura 5.39). La función que cumple el botón representado en la Figura 5.40, es ordenar al microposicionador que realice una inicialización del carril. La única manera que existía en el diseño previo para realizarlo, era reiniciando el microcontrolador y dejando presionado unos segundos el pulsador 5 (representado

con el símbolo ↵). Por ello, al incluir este botón, se evita tener que reiniciar el microcontrolador y perder su conexión con el PC a través del programa LabVIEW.

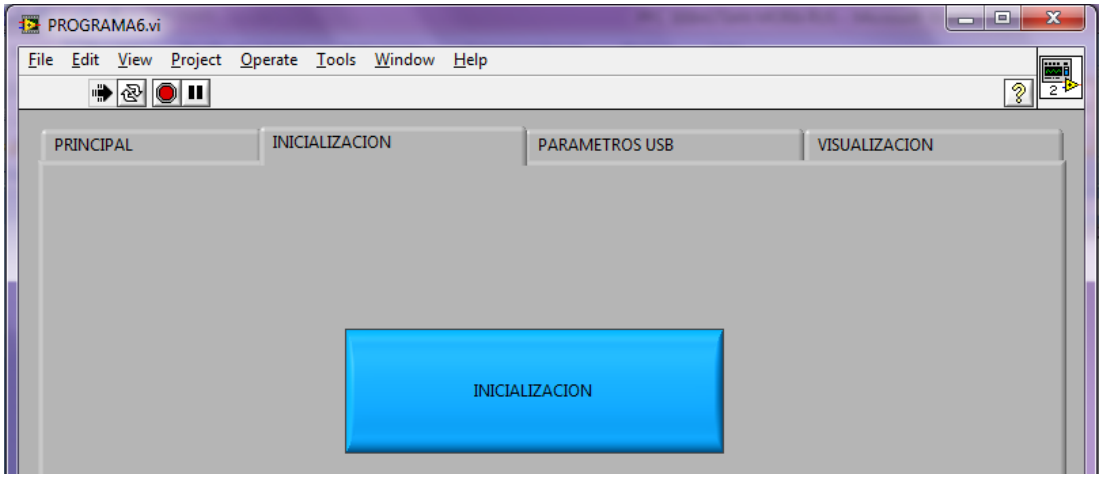


Figura 5.39 Pestaña secundaria INICIALIZACIÓN

Figura 5.40 Botón de inicialización del carril

Pestaña “PARAMETROS USB”

En esta nueva pestaña llamada “PARAMETROS USB” están incluidos todos los parámetros necesarios para que el puerto USB funcione correctamente (ver Figura 5.41).

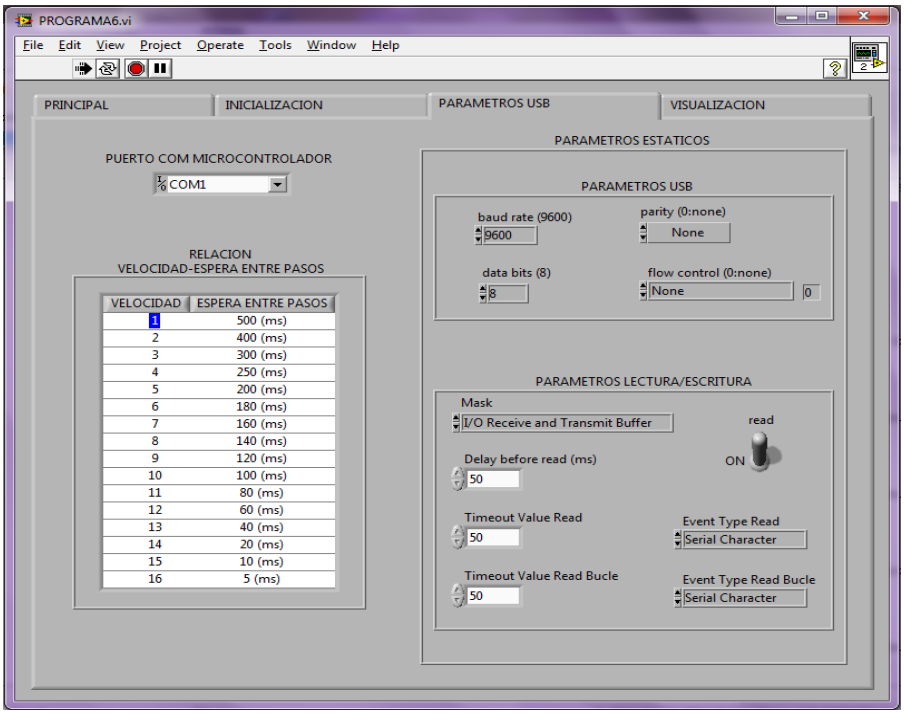


Figura 5.41 Pestaña secundaria "PARAMETROS USB"

El único parámetro que podrá ser configurado por el usuario es el llamado "PUERTO COM MICROCONTROLADOR". Con él se seleccionará el puerto COM por el que se ha conectado el microposicionador al PC mediante un menú desplegable (ver Figura 5.42), como ya se detalló previamente.

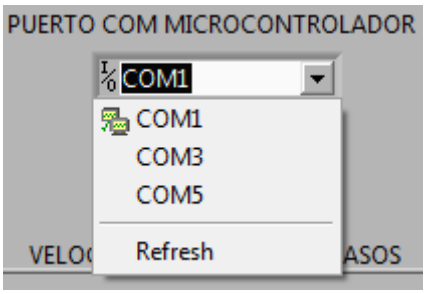


Figura 5.42 Selección de puerto COM

El resto de parámetros de la pestaña "PARAMETROS USB" están incluidos en un recuadro con el nombre "parámetros estáticos" y los campos a los que se refieren aparecen, en su mayoría, en color gris. Esta nomenclatura indica que dichos parámetros no deben ser modificados por un usuario inexperto (ver Figura 5.43). Entre el conjunto de parámetros estáticos existen dos tipos diferenciados:

1. "PARAMETROS USB" ó serie: Están incluidos todos los parámetros de configuración del puerto serie descritos previamente en el apartado 3.3.3.

2. “PARÁMETROS LECTURA/ESCRITURA”: Son los encargados de configurar la lectura y la escritura del programa LabVIEW en el puerto serie.

   - *Mask*: Habilita un buffer de lectura, escritura o ambos. Por defecto, está configurado para que habilite un buffer de lectura y otro de escritura de un tamaño de 5 *bytes*.
   - *Delay before read*: Se encarga de esperar el tiempo introducido en el campo correspondiente antes de proceder a leer del puerto serie. El aumento de este parámetro podría provocar la pérdida de información si los datos son enviados a una frecuencia muy alta.
   - *Timeout Value Read*: El valor que se indica en este campo es el tiempo que está esperando la función de escritura para leer un dato. Si en ese periodo no recibe nada, se pasará a ejecutar otra función.
   - *Timeout Value Read Bucle*: Tiene el mismo significado que el campo anterior, pero que en este caso se utiliza para la lectura cuando se ejecuta el bucle.
   - *Read*: Este interruptor habilita/deshabilita la recepción de datos por el puerto serie.
   - *Event Type Read*: Habilita la lectura de un tipo concreto de interrupción (evento). Estos pueden ser de diferente índole: *Serial character*, *USB interrupt*, *trigger*, etc.
   - *Event Type Read Bucle*: Suspende la ejecución del subproceso donde esté conectado hasta que el tipo de evento seleccionado (en este caso, *Serial character*) se ejecuta. Es decir, en este caso se suspenderá el subproceso que se está ejecutando hasta que se lea del puerto serie.

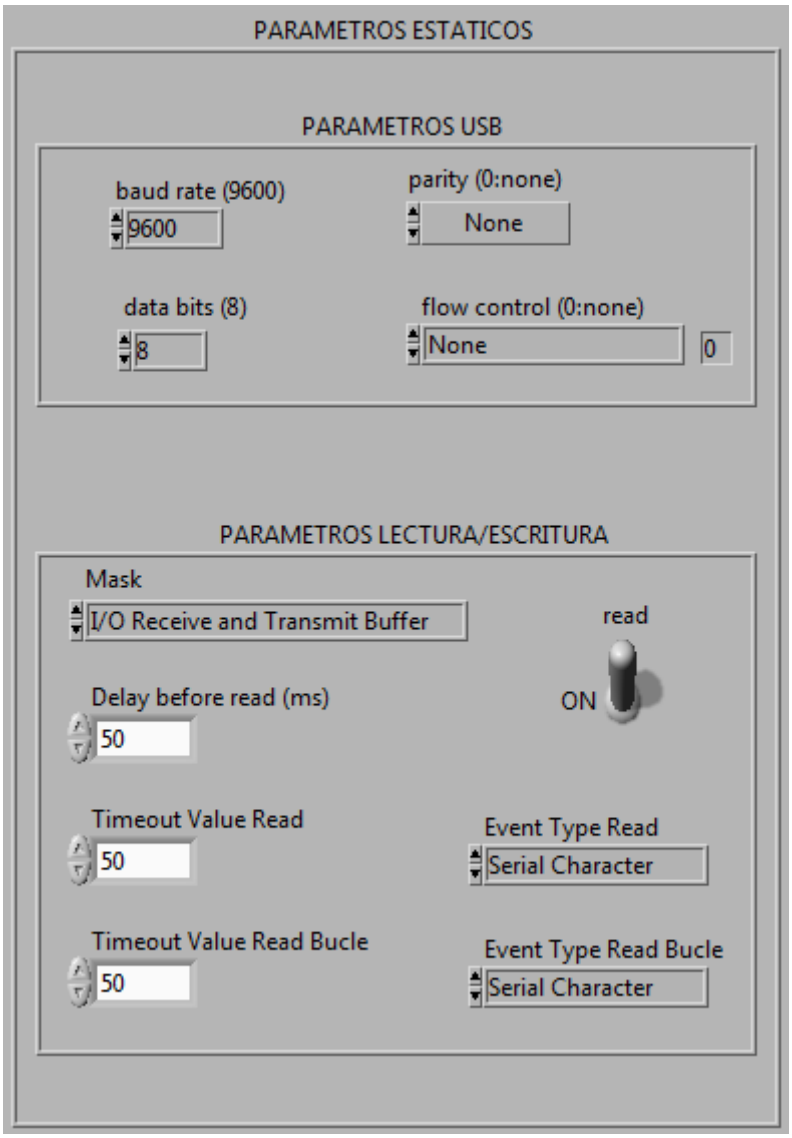
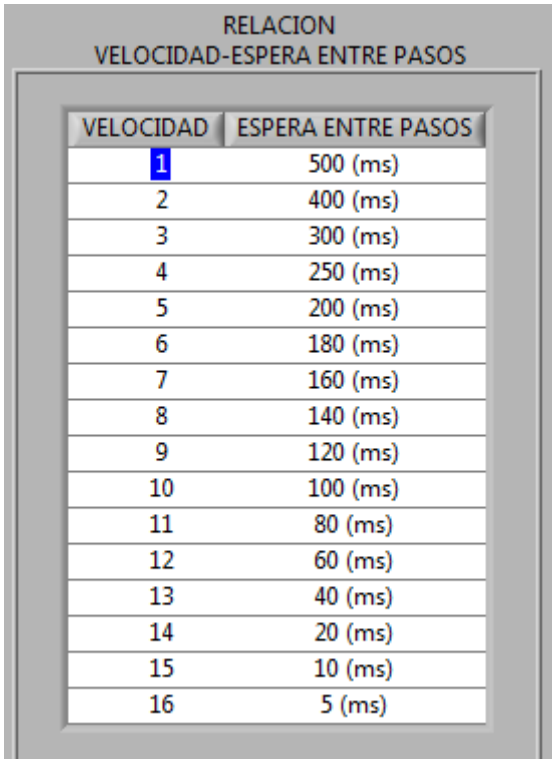


Figura 5.43 Parámetros estáticos del USB

Para finalizar con la descripción de la pestaña “PARAMETROS USB”, se incluye una tabla que se ha representado en la Figura 5.44. En la tabla se muestra el listado que relaciona cada una las velocidades de movimiento del motor paso a paso, numeradas del 1 al 16, con el tiempo real de espera entre la ejecución de dos pasos consecutivos para cada caso. Esta tabla no está relacionada con los parámetros USB, sin embargo se ha considerado adecuado su inclusión en esta pestaña para que el usuario final la visualice cada vez que se abre el programa. Se recuerda que, por defecto, al abrir el programa aparece activa la pestaña “PARAMETROS USB”. Además, el usuario podrá consultar el listado posteriormente cuando lo desee.



VELOCIDAD	ESPERA ENTRE PASOS
1	500 (ms)
2	400 (ms)
3	300 (ms)
4	250 (ms)
5	200 (ms)
6	180 (ms)
7	160 (ms)
8	140 (ms)
9	120 (ms)
10	100 (ms)
11	80 (ms)
12	60 (ms)
13	40 (ms)
14	20 (ms)
15	10 (ms)
16	5 (ms)

Figura 5.44 Tabla de "RELACIÓN VELOCIDAD-ESPERA ENTRE PASOS"

La cuarta, y última pestaña, "VISUALIZACIÓN", será descrita en el apartado 5.3.1 puesto que fue diseñada para supervisar el intercambio de información entre los dispositivos y no como pestaña configurable.

5.2.3 MENSAJES DE ADVERTENCIA

En este apartado se muestran todas las evaluaciones que se han llevado a cabo para validar el correcto envío y la correcta recepción de información entre el microcontrolador y el PC a través del programa LabVIEW.

La primera y más importante evaluación que se llevó a cabo, fue la tarea de discriminar si, de los 5 *bytes* recibidos, el primero y el último son, o no, coherentes con el inicio y final de trama definidos en el protocolo de comunicación. Si esos *bytes* son correctos, el programa pasa a ejecutar las acciones correspondientes. En cambio, si por cualquier motivo alguno de estos dos *bytes* se recibe mal o es erróneo, aparecerá un mensaje de error para informar al usuario (ver Figura 5.45). Hasta la fecha no se han dado errores en la recepción de la trama, pero se ha incluido la verificación como medida de protección. El error mostrado en la Figura 5.45 ha sido generado única y exclusivamente para verificar su correcto funcionamiento.

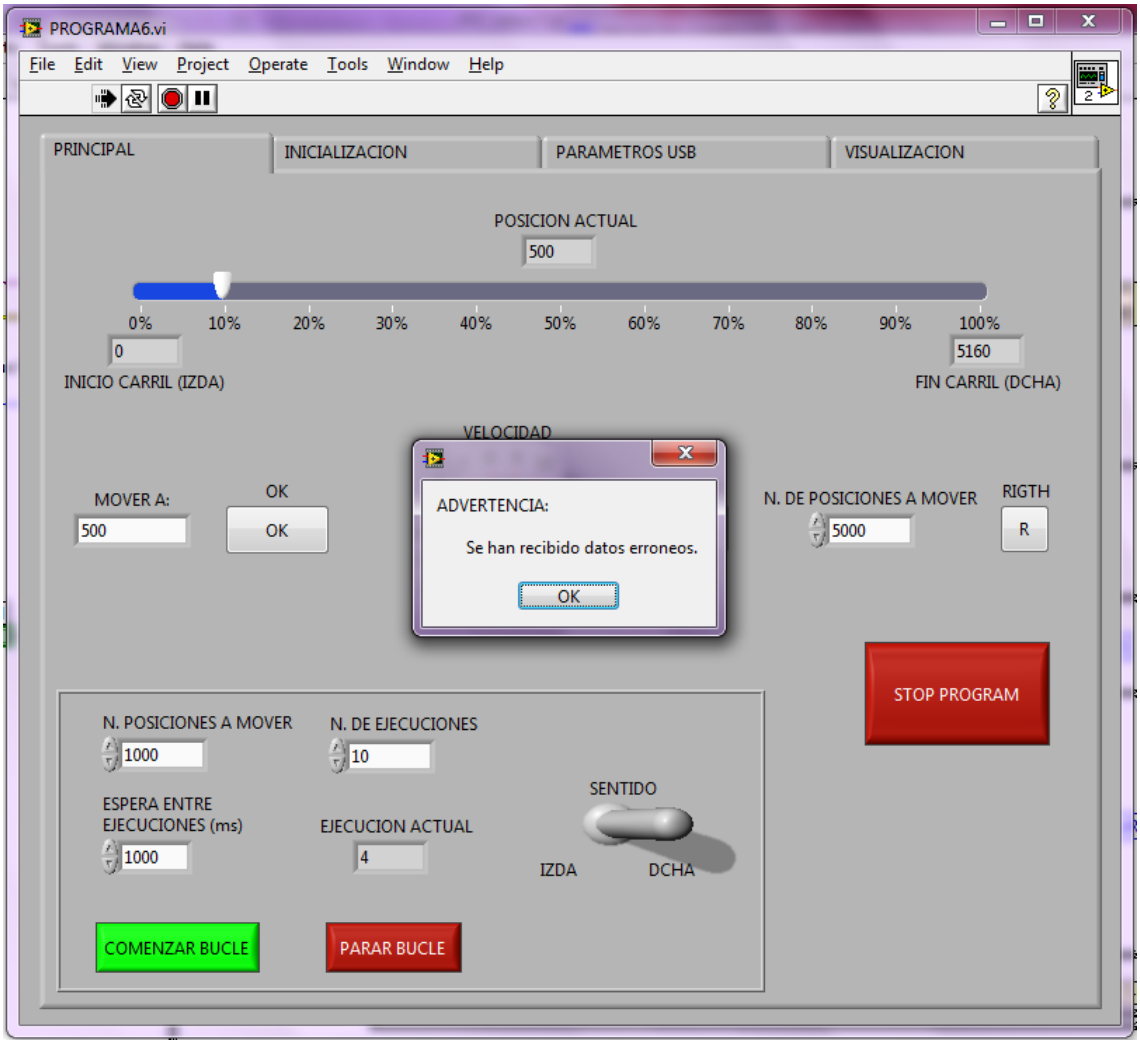


Figura 5.45 Evaluación de recepción correcta de datos y mensaje de error

Otro aspecto que se ha tenido en cuenta, tiene que ver con la definición de posiciones de destino de movimiento enviadas al microcontrolador, que no están contempladas dentro del rango del carril. Esta medida de seguridad evita que el usuario esté pendiente de verificar constantemente que se está moviendo dentro del rango útil del carril. Como ya se comentó, esta protección también fue implementada de forma local con el microcontrolador y ahora se refuerza a nivel de LabVIEW. Para evaluar esta situación se han estudiado y abordado todos los casos en los que puede aparecer.

La primera verificación se realiza con la introducción de los datos de partida en los campos. En cualquiera de ellos no es válido introducir valores menores que 1, puesto que no se trabajará con 0 posiciones, ni con pasos negativos. En segundo lugar se comprueba que en el visualizador “MOVER A:” no se introduzca un valor que supere al final de carril derecho ni izquierdo mostrándose, en caso contrario, un mensaje de

error como en el ejemplo de la Figura 5.46 (el carril acaba en la posición 5165, por tanto, la posición destino no puede ser la 5500.

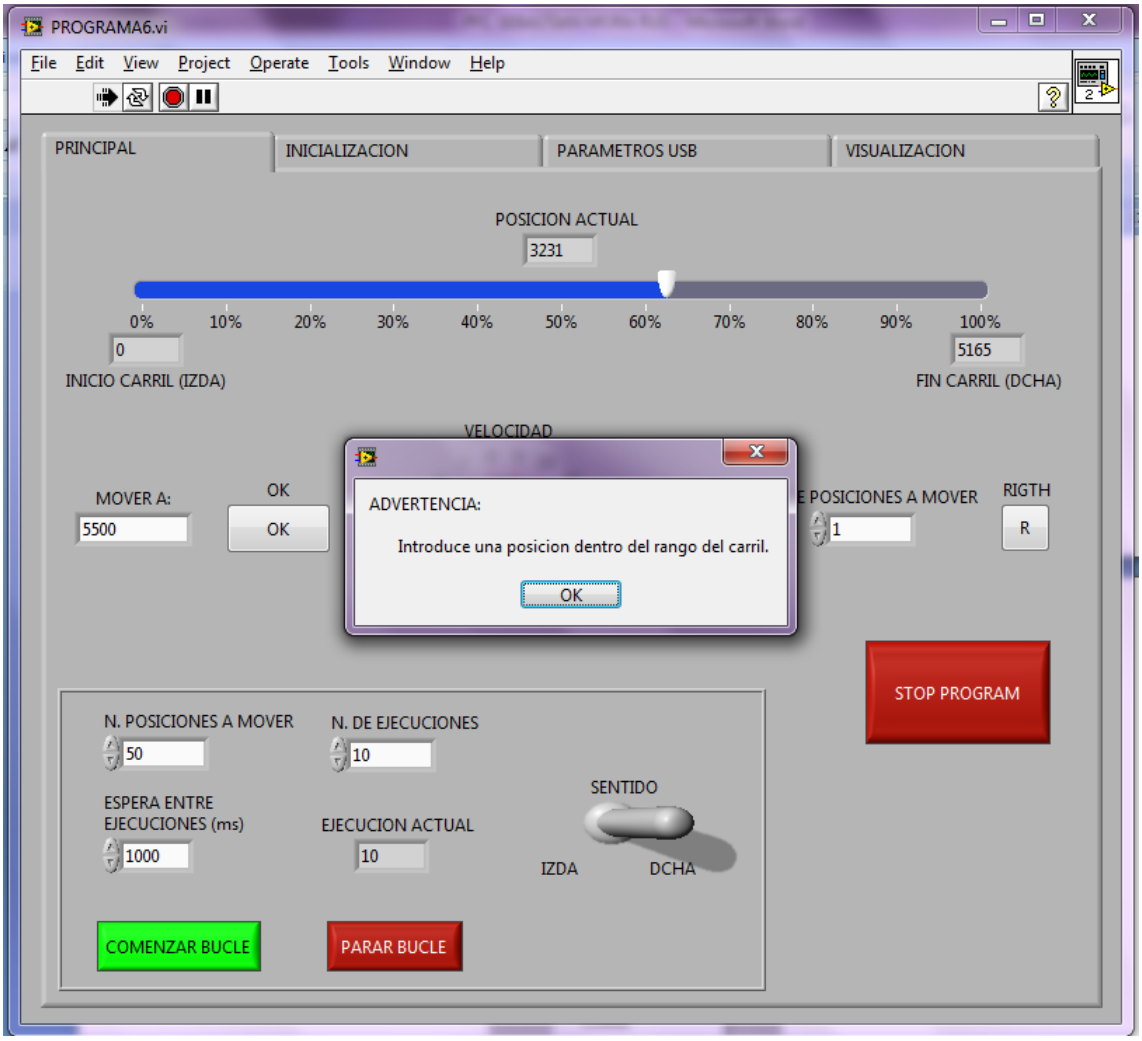


Figura 5.46 Advertencia de seguridad en movimientos largos fuera del rango del carril

Por otro lado, también se comprueba que para recorridos cortos de la vagoneta el usuario tampoco pueda definir posiciones de destino incoherentes. Si se quiere mover la vagoneta un número de posiciones que sobrepasa el extremo derecho, el mensaje de advertencia que mostrará el programa LabVIEW viene representado en la Figura 5.47. En el ejemplo, la posición actual de la vagoneta es la 3231 y hay que avanzar a la derecha 2000 pasos, dando lugar a una posición de destino fuera del carril.

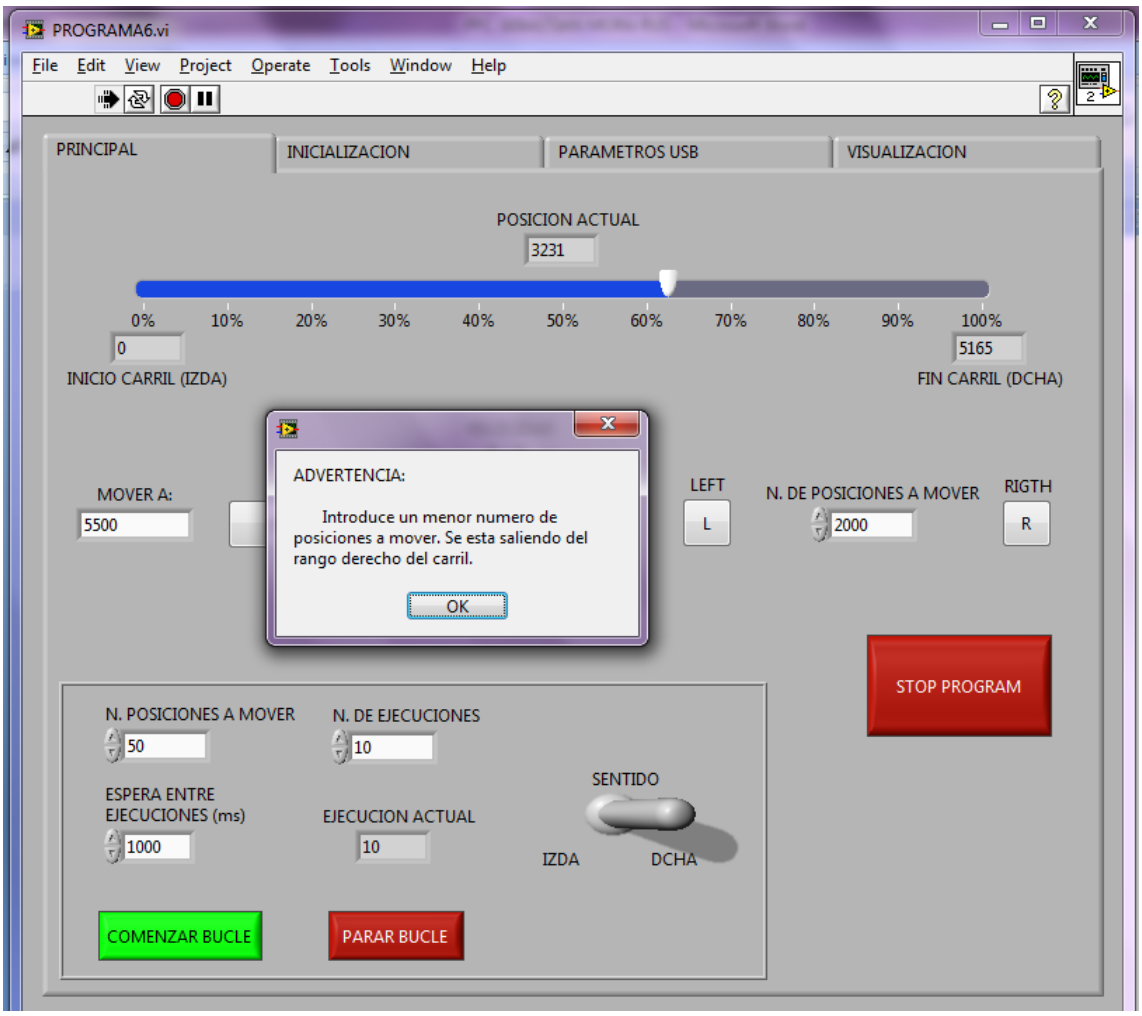


Figura 5.47 Advertencia de seguridad en movimientos cortos hacia la derecha fuera del rango del carril

Si, por el contrario, se desea mover la vagoneta un número de posiciones hacia la izquierda mayor que la posición actual que ocupa en el carril, el usuario estaría indicando al LabVIEW que quiere alcanzar posiciones negativas en el carril, situación que no puede darse. En ese caso se mostrará la advertencia que aparece en la Figura 5.48. En el ejemplo, la posición actual que ocupa la vagoneta es la 500. Si se desea mover 550 pasos hacia la izquierda, la posición de destino estará fuera del carril.

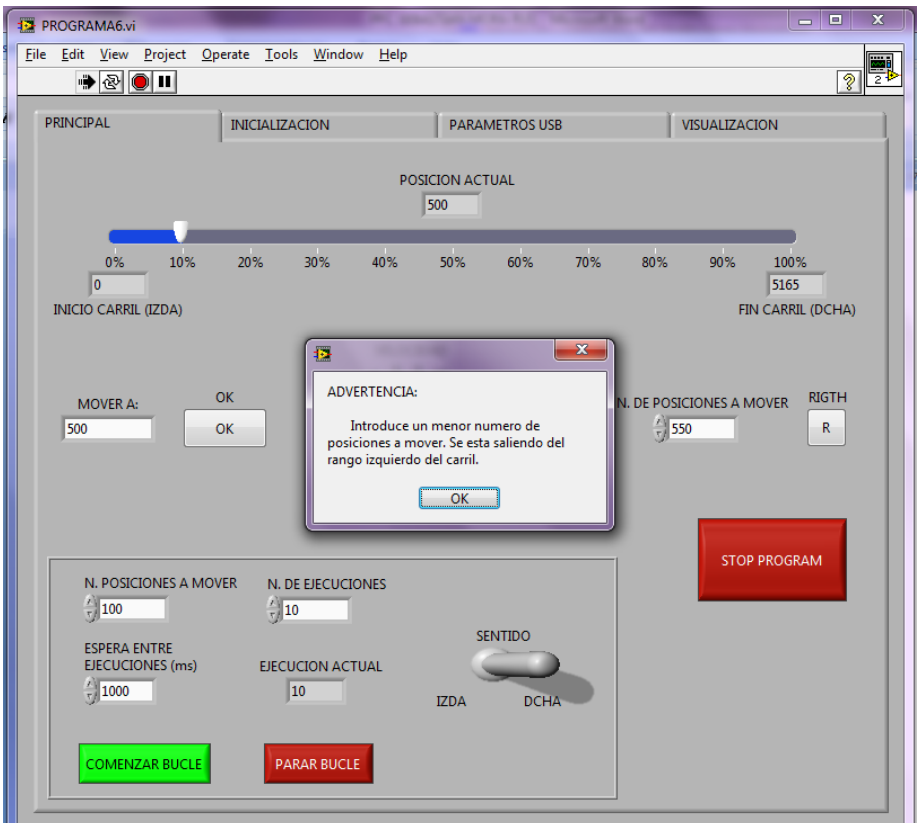


Figura 5.48 Advertencia de seguridad en movimientos cortos hacia la izquierda

En último lugar, el programa comprobará también que durante la ejecución del bucle no se incurre en alguno de los problemas previos que se han comentado. Es por ello que se calcula la última posición a la que irá la vagoneta, y si se sale fuera del rango del carril, mostrará un mensaje de advertencia como el de la Figura 5.49. En el ejemplo, la posición actual es la 4200. El bucle define un avance hacia la derecha de 1000 pasos, es decir, 10 ejecuciones de 100 pasos cada una. Con estos parámetros, la posición de destino sería la 5200, provocando la aparición del mensaje de advertencia comentado.

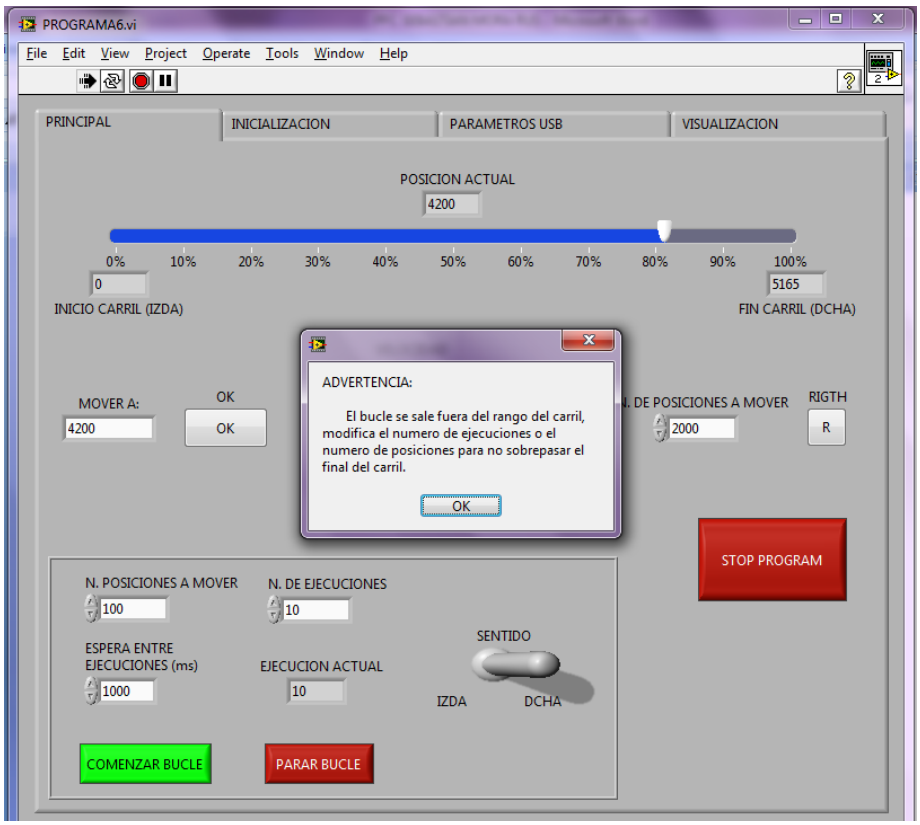


Figura 5.49 Advertencia de seguridad en movimientos que conllevan ejecución del bucle

5.3 COMUNICACIÓN PC Y MICROCONTROLADOR

Como ya se comentó, el protocolo de comunicación y la correcta sincronización entre los dos sistemas, PC y microcontrolador, es uno de los aspectos más importante del presente proyecto. Por tanto, en este apartado se hablará de los métodos de que dispone el usuario para comprobar que la comunicación entre los dispositivos se está realizando adecuadamente. La comprobación será fundamentalmente por visualización por parte del usuario de los indicadores de la aplicación. También se detallará en este apartado un caso muy específico relacionado con la desconexión involuntaria de equipos y los pasos a seguir una vez que se ha verificado que no existe ningún tipo de sincronización entre ellos.

5.3.1 VERIFICACION DE FUNCIONAMIENTO DE LA APLICACIÓN

La comprobación de funcionamiento de la aplicación, que verificará que la aplicación global funciona correctamente, se puede realizar por dos vías.

Una de ellas, consiste en observar los visualizadores de la pestaña “PRINCIPAL”. Estos campos deben cambiar pasados unos segundos desde que la aplicación es ejecutada. Si desde el primer momento, los campos de esta pestaña se actualizan, se asegura que la aplicación funciona con total normalidad. En el presente documento, ya se han ido incluyendo imágenes de estos visualizadores (a partir del apartado 5.2.2) y de los valores adecuado para asegurar el correcto funcionamiento.

Aparte de la comprobación visual por parte del usuario de la pestaña “PRINCIPAL”, se ha incluido una última pestaña llamada “VISUALIZACION” (ver Figura 5.50) con el fin de seguir la ejecución de la escritura y la lectura de información, a nivel de los *bytes* de la trama del protocolo de comunicación entre dispositivos. En dicha pestaña se pueden visualizar los parámetros internos más importantes que no son mostrados en otras pestañas.

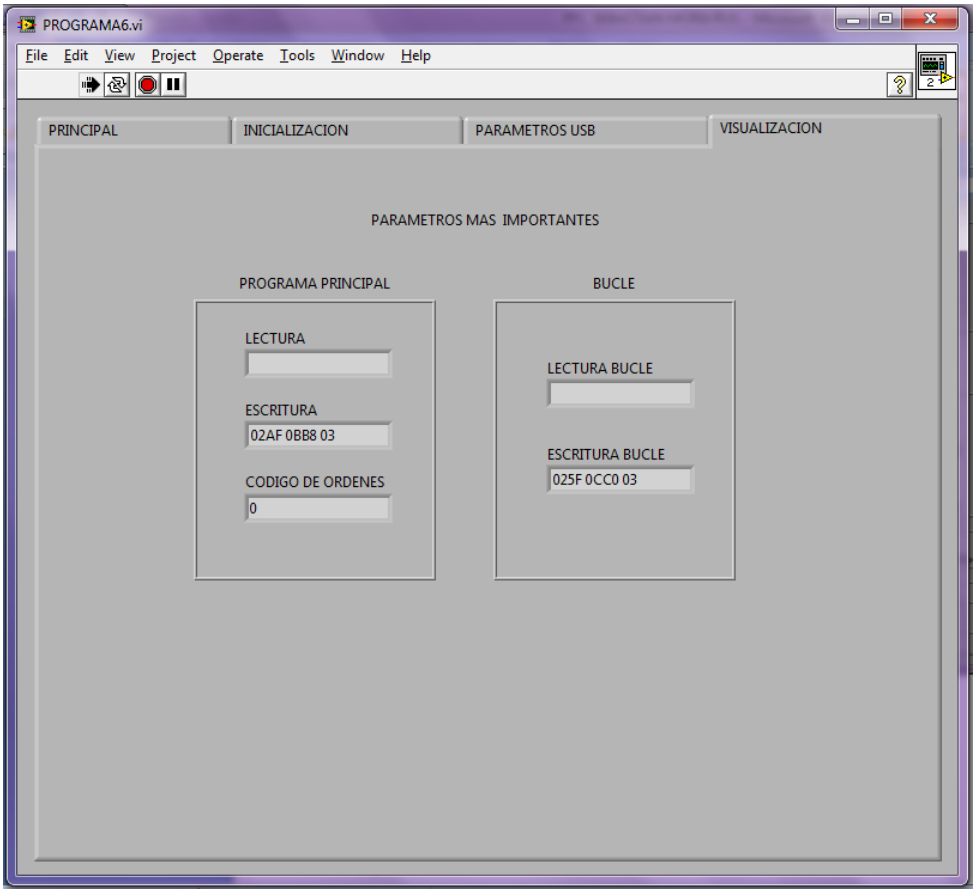


Figura 5.50 Pestaña de la aplicación “VISUALIZACIÓN”

Tal como se observa en la Figura 5.50 los parámetros se han dividido en dos grupos. El primero grupo se corresponde con los parámetros del programa principal (Lectura, Escritura y Código de órdenes) y el segundo está relacionado con los parámetros de la ejecución del bucle. Esto se hace así para no mezclar los parámetros de un proceso y

de otro, puesto que la escritura y lectura del puerto serie se realiza de manera independiente cuando se ejecuta el programa principal, o se lleva a cabo el bucle repetitivo.

Se ha decidido mostrar la trama de 5 *bytes*, es decir, un total de 40 bits en binario, en formato hexadecimal, que equivale a 10 dígitos. (Recordar que 8 *bits* (*1 byte*) equivalen a 2 dígitos en formato hexadecimal). De esta manera es más sencillo visualizar 10 cifras que visualizar 40. Y solamente con realizar un cambio sencillo, se puede pasar de hexadecimal a binario. En la Figura 5.51 se observa un ejemplo en el que el programa LabVIEW ha leído 5 *bytes* procedentes del microcontrolador (visualizador "LECTURA"). Se observa que dichos *bytes* llevan un orden correcto que coincide con lo esperado según el protocolo de comunicación. Es decir, con el visualizador "LECTURA" se comprueba que verdaderamente la comunicación es correcta cuando el microcontrolador envía información al PC. Veámoslo a continuación.

- El primer *byte* es el hexadecimal 0x02 cuyo equivalente al binario sería el 0b00000010 [5] y que coincide con el STX (inicio de trama, *Start of text*) del protocolo de comunicación.

- El segundo *byte* leído de la trama se corresponde con el código de acción. Este *byte* se ha extraído de la trama y se ha representado individualmente en el visualizador "CODIGO DE ORDENES". En el ejemplo se trata del numero hexadecimal 0x00 que, indica al programa LabVIEW que los siguientes 2 *bytes* son la posición en la que se encuentra la vagoneta. (ver capitulo 3.3.2)

- Por último, el *byte* que se recibe en quinto lugar es el hexadecimal 0x03. Transformando este número a binario se convierte en 0b00000011, que coincide con el ETX (fin de trama, *End of text*) del protocolo de comunicación.

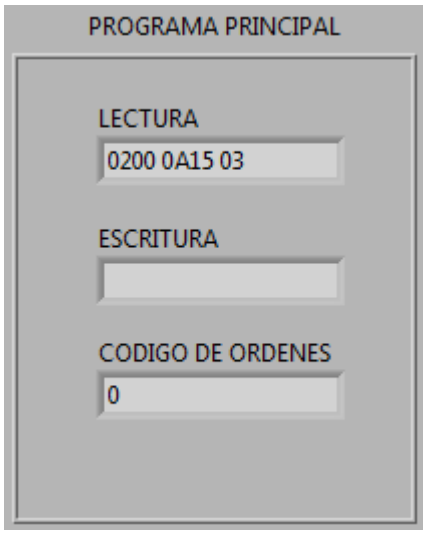


Figura 5.51 Visualización de lectura del puerto serie y del código de ordenes recibido

A continuación se testeará que el envío de información del LabVIEW al microcontrolador (“ESCRITURA”), es correcta. Hasta ahora, se ha confirmado que el programa LabVIEW lee perfectamente los datos recibidos del microcontrolador. En la Figura 5.52 se observa que verdaderamente los datos enviados al microposicionador coinciden también con el protocolo desarrollado. A continuación (ejemplo) se detallan los datos enviados “ESCRITURA”:

- El primer *byte* es el hexadecimal 0x02 cuyo equivalente al binario sería el 0b00000010 [5] y que coincide con el STX (inicio de trama, *Start of text*) del protocolo de comunicación.
- El segundo *byte* que se envía al microposicionador se corresponde con el código de acción. En el ejemplo se trata del numero hexadecimal 0x5F que al transformarlo a binario nos queda el valor 0b0101111. Observando la codificación en el capitulo 3.3.2, comprobamos que las ordenes enviadas al microcontrolador corresponden con sentido de movimiento a derechas (0101|XXXX) y una velocidad de 16 (XXXX|1111) que equivale a la velocidad máxima.
- Los dos siguientes *bytes* (0x0B y 0xB8), indican la posición destino a la que debe ir la vagoneta, que según el código de órdenes debe estar a la derecha de la posición actual, y cuyo valor equivale a la posición 3000 (0b0000101110111000).

- Por último, el *byte* que se envia en quinto lugar es el hexadecimal 0x03. Transformando este número a binario se convierte en 0b00000011, que coincide con el ETX (fin de trama, *End of text*) del protocolo de comunicación.

En la pestaña “VISUALIZADOR”, siempre se muestra la última orden recibida por parte del microcontrolador “CODIGO DE ORDENES”. En el ejemplo de la Figura 5.52 se observa que el último valor recibido corresponde con el 0xB3 en hexadecimal. Realizando la conversión a binario, se obtiene el número 0b10110011 que se corresponde (ver capitulo 3.3.2) con el recibo del final del extremo derecho.

Figura 3.52 Visualización de escritura en el puerto serie y último código de órdenes recibido

Por último, se verificará que la lectura y la escritura que realiza el bucle repetitivo es la correcta. En la Figura 5.53, se muestra el ejemplo de una trama que ha sido enviada del LabVIEW al microcontrolador (“ESCRITURA”), y posteriormente ha reenviado el microcontrolador al LabVIEW (“ESCRITURA”). Esto es así porque, al ejecutar el bucle, primeramente se envía al microcontrolador la posición destino, y una vez que la vagoneta se posiciona, informa al LabVIEW de su posición actual, para así ejecutar otro ciclo del bucle (ver capitulo 3.3.6). A continuación se describe la información del visualizador “ESCRITURA”:

- El primer *byte* es el hexadecimal 0x02 cuyo equivalente al binario sería el 0b00000010 [5] y que coincide con el STX (inicio de trama, *Start of text*) del protocolo de comunicación.

- El segundo *byte* que se envía al microposicionador se corresponde con el código de acción. En el ejemplo se trata del numero hexadecimal 0x5F que al transformarlo a binario nos queda el valor 0b0101111. Observando la codificación en el capitulo 3.3.2, comprobamos que las ordenes enviadas al microcontrolador corresponden con sentido de movimiento a derechas (0101|XXXX) y una velocidad de 16 (XXXX|1111) que equivale a la velocidad máxima.

- Los dos siguientes *bytes* (0x0B y 0x0F), indican la posición destino a la que debe ir la vagoneta, que según el código de órdenes debe estar a la derecha de la posición actual, y cuyo valor equivale a la posición 2831 (0b0000101100001111).

- Por último, el *byte* que se envía en quinto lugar es el hexadecimal 0x03. Transformando este número a binario se convierte en 0b00000011, que coincide con el ETX (fin de trama, *End of text*) del protocolo de comunicación.

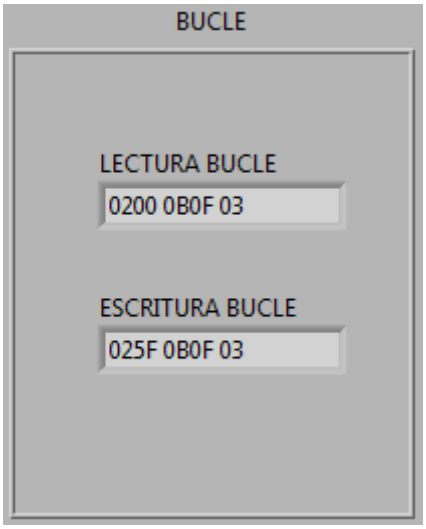


Figura 5.53 Visualización de escritura y posterior lectura del bucle en el puerto serie

A continuación se describe la información del visualizador “LECTURA”:

- El primer *byte* es el hexadecimal 0x02 cuyo equivalente al binario sería el 0b00000010 [5] y que coincide con el STX (inicio de trama, *Start of text*) del protocolo de comunicación.

- El segundo *byte* leído de la trama se corresponde con el código de acción. En el ejemplo se trata del numero hexadecimal 0x00 que, indica al programa LabVIEW que los siguientes 2 *bytes* son la posición en la que se encuentra la vagoneta. (ver capitulo 3.3.2)

- Los dos siguientes *bytes* (0x0B y 0x0F), se corresponden con la posición descrita en el bucle en el visualizador “ESCRITURA” (2831).

- Por último, el *byte* que se envía en quinto lugar es el hexadecimal 0x03. Transformando este número a binario se convierte en 0b00000011, que coincide con el ETX (fin de trama, *End of text*) del protocolo de comunicación.

5.3.2 DESCONEXION INVOLUNTARIA ENTRE DISPOSITIVOS

Existen varios tipos de desconexiones involuntarias que pueden acarrear que la comunicación futura entre los dos dispositivos no se realice correctamente. Es por ello que a continuación se describen los pasos a seguir en función de cada tipo de desconexión.

DESCONEXION DEL PROGRAMA LABVIEW DE MANERA CORRECTA

Este primer tipo de desconexión es la más común, y se produce cuando el usuario decide parar el programa LabVIEW de una manera correcta (Pulsando el botón STOP). Puede venir dada por un motivo de finalización del trabajo, porque el usuario se ha despistado, o simplemente porque ha decidido parar la aplicación un momento para volver a retomarla posteriormente.
En este caso, los pasos a seguir para volver a utilizar el programa son los siguientes:

1. Desconectar el cable USB del extremo del microposicionador ó del PC.
2. Volver a conectar el cable USB en el puerto en que se encontraba conectado.
3. Volver a ejecutar la aplicación.

DESCONEXION DE LA COMUNICACIÓN SIN PARAR EL PROGRAMA LABVIEW

Se produce este tipo de desconexión, cuando el usuario desconecta el cable USB por uno de sus extremos, o realiza un RESET al microposicionador, sin antes haber parado la aplicación de forma correcta.

Para este caso, los pasos que hay que seguir para volver a utilizar el programa son los siguientes:

1. Parar el programa de manera correcta (cliqueando en el botón STOP PROGRAM) y aceptar los posibles mensajes de error de conexión si se muestran (ver Figura 5.54).
2. Desconectar y volver a conectar el cable USB.
3. Volver a ejecutar el programa.

Si después de ejecutar el programa, sigue sin haber comunicación, realizar los pasos del apartado “DESCONEXION DEL PROGRAMA LABVIEW SIN PULSAR STOP PROGRAM”, que se detalla a continuación.

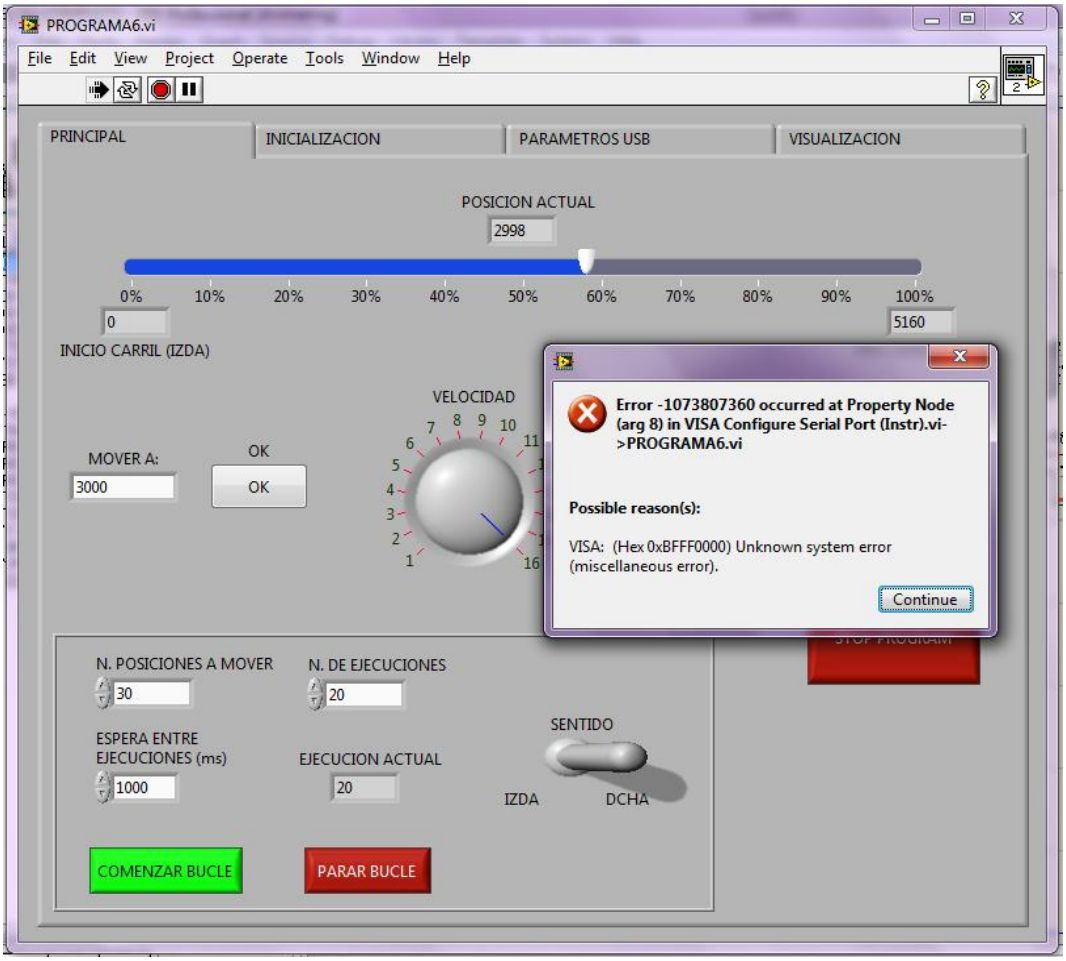


Figura 5.54 Mensaje de error cuando no se cierra correctamente el puerto serie

El mensaje de error mostrado en las Figura 5.54 viene producido por una desconexión repentina de la comunicación entre los dispositivos, provocando que el puerto serie no se cierre correctamente y se incida en errores internos que solamente pueden ser subsanados, conectando de nuevo la aplicación, y cerrándola adecuadamente.

DESCONEXION DEL PROGRAMA LABVIEW SIN PULSAR STOP PROGRAM

Si el usuario final para la aplicación de una manera incorrecta, los pasos a seguir serán los siguientes:

1. Ejecutar el programa.
2. Pararlo con el botón de STOP aceptando si se muestra el mensaje de error de la Figura 5.54.
3. Desconectar y volverá conectar el cable USB.
4. Volver a ejecutar la aplicación.

Es preciso recalcar que el usuario final deberá seguir esta secuencia de forma completa y ordenada. En caso de omitir los dos primeros pasos, la aplicación no funcionará y tendrá, finalmente, que empezar de nuevo.

CAPÍTULO 6

CONCLUSIONES Y TRABAJOS FUTUROS

6.1 CONCLUSIONES

Una vez realizado el proyecto y hechas las comprobaciones de funcionamiento correspondientes, se ha llegado a las siguientes conclusiones. Se ha logrado realizar las mejoras mecánicas que se plantearon, y con ello se ha mejorando la repetitividad del ancho del carril, la conexión y desconexión del sistema hardware y la facilidad de reconocer fácilmente la función asociada a cada pulsador por parte del usuario. También se han cumplido todos los objetivos referentes al software del microcontrolador, dotándolo de un programa modular, un protocolo de comunicación y una serie de mejoras de inicialización de carril, lectura de pulsadores, sistema de seguridad de finales de carril y visualizaciones de la pantalla LCD.

Por otro lado, al abordar los objetivos del LabVIEW se han conseguido superar todos los puntos mencionados, resaltando el protocolo de comunicación, el monitoreo completo del microposicionador y el sistema automático de posicionamiento de la vagoneta mediante bucles repetitivos.

HARDWARE

- ✓ Se ha construido un sistema de anclaje alternativo para los finales de carrera al ya existente.
- ✓ Se ha sustituido el botón de reset por uno de superficie.
- ✓ Se han incluido pegatinas informativas que muestran la función que desempeña cada pulsador.
- ✓ Se ha dotado a los finales de carril y al motor de un conexionado rápido y sencillo.

SOFTWARE

- ✓ Se ha hecho una reorganización del *software* del microcontrolador dotándolo de funciones modulares que ayudarán al usuario en ampliaciones futuras.

- ✓ Se ha programado una inicialización (Barrido del ancho del carril total), guardando la posición de ambos extremos para su utilización posterior.

- ✓ Se ha conseguido posicionar la vagoneta en la mitad del carril, una vez que se realiza una inicialización del carril.

- ✓ Se ha logrado acotar el ancho del carril a gusto del usuario cuando se ejecuta una inicialización.

- ✓ Se ha mejorado el sistema de visualización LCD, evitando el parpadeo del mismo.

- ✓ Se ha programado la visualización en todo momento de la búsqueda en movimientos largos.

- ✓ Se ha aumentado la velocidad de búsqueda en movimientos largos.

- ✓ Se ha diseñado un protocolo de comunicación para comunicar ambos aparatos (Labview y microcontrolador).

- ✓ Dotarlo de un sistema de seguridad con los siguientes puntos:

    - Si por cualquier motivo, el usuario desea mover la vagoneta en un sentido y su final de carril está pulsado, imposibilidad de desplazamiento.

    - Si la vagoneta se está moviendo y por cualquier motivo el final de carril hacia donde se desplaza es pulsado, la vagoneta debe pararse inmediatamente sin repercutir de manera negativa en ningún parámetro.

LABVIEW

Se ha realizado un *software* de control para el microposicionador que ha incluido los siguientes puntos:

- ✓ Se ha diseñado un protocolo de comunicación capaz de comunicarse con el microposicionador.

- ✓ Se ha monitoreado en tiempo real la posición de la vagoneta y de sus finales de carril.

- ✓ Se ha conseguido enviar al microcontrolador la posición destino solamente pulsando un botón.

- ✓ Se ha logrado desplazar a la vagoneta (en ambos sentidos) un número específico de posiciones sin que el usuario calcule la posición destino.

- ✓ Se ha incluido un botón con el que se indica la velocidad de movimiento que llevará la vagoneta.

- ✓ Se ha dotado a la aplicación de un botón de inicialización de carril para no tener que reiniciar el microposicionador cada vez que se quiera ejecutar esta acción.

- ✓ Se ha otorgado a la aplicación de un sistema automático de posicionamiento de la vagoneta mediante bucles repetitivos, introduciendo solo 3 parámetros de control.

- ✓ Se ha conseguido incluir visualizadores para monitorear los datos enviados y recibidos por el USB.

6.2 TRABAJOS FUTUROS

Antes de abordar las mejoras futuras, hay que remarcar que, una vez desarrollada la aplicación en entorno gráfico, se ha creado un fichero ejecutable para poder instalar el programa desarrollado en cualquier ordenador que no disponga del programa LabVIEW. A partir de la aplicación creada, se pueden plantear algunas mejoras futuras en el dispositivo. A continuación se listan algunas de las propuestas planteadas:

1- Algunas mejoras relativas a la parte mecánica del sistema pueden ser la sustitución del motor actual por uno más preciso y el uso de un tornillo sin fin para evitar el reajuste obligado por el destensado de la correa. También se

puede dotar al sistema de sensores de infrarrojos para conocer la posición exacta en cada momento de la vagoneta y/o como finales de carrera.

2- Puesto que los dos sistemas se conectan a través de USB, el *driver* del microcontrolador podría ser modificado (DLL) para asignarle un identificador estático y que, de esto modo, el programa LabVIEW lo reconociese automáticamente.

3- Otra mejora sería el envió de la espera entre pasos (velocidad) con dos *bytes* (para tiempos de hasta 65535 milisegundos) para que el usuario pudiera definirlo a merced, puesto que actualmente el tiempo está predefinido con un conjunto de valores discretos.

4- Mejorar la aplicación para que trabaje con señales externas de *trigger* y así sincronizar el programa con eventos externos como la captura con cámaras o la aplicación de señales adecuadas de conmutación a los dispositivos electroópticos que se desea caracterizar.

5- Ampliar la aplicación y trabajar con ficheros que sirvan para guardar o leer posiciones concretas de la vagoneta.

BIBLIOGRAFIA

REFERENCIAS BIBLIOGRÁFICAS

1. ***"Sistema microposicionador motorizado en un eje para caracterización de dispositivos electroópticos",*** Proyecto Fin de Carrera de Jorge Gómez Marcos (Julio 2011).

2. ***"LabVIEW For Everyone Third Edition"***, Jeffrey Travis, Jim Kring, Ed. Prentice hall. (Noviembre 2010).

3. ***LabVIEW™ Help***, © 2005–2010 National Instruments Corporation. All rights reserved (Noviembre 2010).

4. ***Foros de discusión***, © 2005–2010 National Instruments Corporation. http://spain.ni.com/ (Noviembre 2010).

5. ***Código de caracteres ASCII*.** http://www.astrohandbook.com/files/ascii_codes.html (Noviembre 2011).

6. **"*C Compiler Reference Manual*"**, Copyright © 1994, 2011. Custom Computer Services, In. http://www.ccsinfo.com/downloads/ccs_c_manual.pdf (Julio 2011).

7. **"*Manual de Usuario del Compilador PCW de CCS*",** C Compiler for Microchip PICmicro MCUs. Escrito por Andrés Cánovas López. Reeditado para formato PDF por Víctor Dorado. http://www.bairesrobotics.com.ar/data/Manual_Compilador_CCS_PICC.pdf (Marzo 2011).

ANEXOS

ANEXO1: CODIGO FUENTE DEL SISTEMA MICROPOSICIONADOR EN LENGUAJE C.

main.c

```
1: //*********************************************************************
2: //
3: //CONEXIÓN USB-LABVIEW
4: //
5: //USB v3.6
6: //
7: //
8: //*********************************************************************
9: #include <./include/18F2550.h>
10: #device ADC=8
11: #fuses HSPLL,NOWDT,NOPROTECT,NOLVP,NODEBUG,USBDIV,PLL5,CPUDIV1,VREGEN
12: #use delay(clock=48000000)
13: #include "include/usb_cdc.h"                    //USB Configuration
14: #include "./include/flex_lcd.c"
15: #include "include/pic18_usb.h"
16: #define EN                    PIN_C0
17: #define FIND                  PIN_A5
18: #define FINI                  PIN_A4
19: #define A1                  PIN_C7
20: #define A2                  PIN_C6
21: #define B1                  PIN_C2
22: #define B2                  PIN_C1
23: #define STX                   2      //INICIO DE TRAMA
24: #define ETX                   3      //FIN DE TRAMA
25:
26: //VARIABLES GLOBALES
27:
28: int8 test1[]="Test LCD 162D   ";
29: int8 test2[]="PIC18F2550  V3.3";
30: int pulsador;//VARIABLE  QUE CONTENDRÁ EL VALOR QUE DEVOLVERÁ LA FUNCION L
EER PULSADORES
31: unsigned int16 var2;
32: unsigned int16 resul,resul1;
33: int paso[4]={0b1100, 0b0110, 0b0011, 0b1001};//SECUENCIA DEL MOTOR PAP
34: unsigned int16 numero_pasos=0;//NUMERO  DE PASOS QUE SE MOVERÁ EL MOTOR
35: unsigned int16 aux2=0;//VARIABLE  QUE ALBERGARÁ LA POSICION ACTUAL
36: unsigned int16 extremo_derecho=0;//VATIABLE  QUE GUARDARÁ EL EXTREMO DEREC
HO
37: unsigned int16 pos=0;//POSICION  FUTURA A LA QUE SE QUIERE IR
38: unsigned int16 poslabview=0;//POSICION  FUTURA A LA QUE SE QUIERE IR DEL L
ABVIEW
39: int8 i=0;
40: int8 modo=0;
                                    //VARIABLE QUE INDICA SI EL USB SE ENCUENT
RA CONECTADO
41: int8 velocidad=5;//VARIABLE  QUE GUARDARA EL RETRASO CON EL QUE SE MOVERA
EL MOTOR(RETRASO MAXIMO 5ms)
42: int8 codigo=1;//ORDENES  QUE SE ENVIARÁN AL LABVIEW
43: unsigned int8 data_L=0;//PARTE  BAJA DE POSICION ACTUAL QUE SE ENVIARÁ AL
LABVIEW
44: unsigned int8 data_H=0;//PARTE  ALTA DE POSICION ACTUAL QUE SE ENVIARÁ AL
LABVIEW
45: unsigned int8 start_labview=0;//BYTE  DE COMIENZO DE TRAMA ESTANDARIZADO E
N CODIGO ASCII
46: unsigned int8 codigo_labview=0;//ORDENES  RECIBIDAS DEL LABVIEW
```

```
47: unsigned int8 data_H_labview=0;//PARTE  ALTA DE POSICION ACTUAL RECIBIDA D
EL LABVIEW
48: unsigned int8 data_L_labview=0;//PARTE  BAJA DE POSICION ACTUAL RECIBIDA D
EL LABVIEW
49: unsigned int8 end_labview=0;//BYTE  DE FIN DE TRAMA ESTANDARIZADO EN CODIG
O ASCII
50:
51:
52: //SUBRUTINA DE ALMACENAMIENTO DATO EN EEPROM
53:
54: void guarda_eeprom(int address, int data){
55:                     write_eeprom (address, data);
56: }
57:
58: //SUBRUTINA PARA IMPRIMIR LA POSICION ACTUAL EN LA LCD
59:
60: void imprimir_posicion_actual(void){
61:                     lcd_putc("\f");
62:                     lcd_gotoxy(1,1);
63:                     printf(lcd_putc,"Pos.actual:%lu", aux2);
64:                     lcd_gotoxy(1,2);
65:                     printf(lcd_putc,"Pulse");
66: }
67:
68: //SUBRUTINA DE LECTURA EN EEPROM DE POSICION ACTUAL Y FINAL DE CARRIL DERE
CHO
69:
70: void posicion_actual(void){
71:                     unsigned int8 aux;
72:
73:                     aux2 = 0x0000;
74:                     aux =read_eeprom (0x02);
75:                     aux2 =  aux<<8;
76:                     aux = read_eeprom (0x01);
77:                     aux2 = aux2 | aux;
78:                     numero_pasos=aux2;
79:                     pos=aux2;
80:                     extremo_derecho=0x0000;
81:                     aux =read_eeprom (0x04);
82:                     extremo_derecho =   aux<<8;
83:                     aux = read_eeprom (0x03);
84:                     extremo_derecho = extremo_derecho | aux;
85: }
86:
87: //SUBRUTINA PARA ENVIAR POSICION ACTUAL AL LABVIEW
88:
89: void envio_posicion_labview (){
90:
91:                 if(modo==1){
92:                     if(usb_enumerated()){
93:
94:
                                data_H   =   aux2>>8;//DESPLAZO  LA PARTE ALTA
DE AUX2 HACIA LA PARTE BAJA Y LUEGO LA GUARDO EN DATA_H PORQUE AL IGUALAR POR
DEFECTO COJEMOS LA PARTE BAJA
95:
                                data_L   =   aux2;//EL  PROPIO MICRO CREA UNA V
ARIABLE INTERMEDIA Y NO BORRA AUX2, DESPUES IGUALAMOS LA PATE BAJA DE AUX2 A D
ATA_L
96:                             codigo=0x00;
```

```
97:
                                printf(usb_cdc_putc, "%c%c%c%c%c",STX,codigo,data_
H,data_L,ETX);//   ENVIO AL LABVIEW LA ORDEN DE LO QUE LE LLEGA POR DATA_H_INTE
RMEDIO Y DATA_L_INTERMEDIO
98:                                delay_ms(50);
99:                            }
100:                        }
101: }
102:
103:
104: //SUBRUTINA DE CONTROL DEL MOTOR PASO A PASO
105:
106: void paso_motor(BYTE speed, BYTE dir, BYTE steps) {
107:                        static BYTE stepper_state = 0;
108:                        BYTE i;
109:
110:                        for(i=0; i<steps; ++i) {
111:                           delay_ms(speed);
112:
113:                           if(dir!='L'){
114:                              stepper_state=(stepper_state+1)&(sizeof(paso)-1);
115:                              if(numero_pasos == 0xFFFF){
116:                                 numero_pasos == 0x00;
117:                              }
118:                              else
119:                                 numero_pasos++;
120:                            }
121:                           else{
122:                              stepper_state=(stepper_state-1)&(sizeof(paso)-1);
123:                              if(numero_pasos == 0x00){
124:                                 numero_pasos == 0xFFFF;
125:                              }
126:                              else
127:                                 numero_pasos--;
128:                           }
129:
130:                           switch(stepper_state){
131:                                  case 0:
132:                                     output_high(A1);
133:                                     output_high(A2);
134:                                     output_low(B1);
135:                                     output_low(B2);;break;
136:                                  case 1:
137:                                     output_low(A1);
138:                                     output_high(A2);
139:                                     output_high(B1);
140:                                     output_low(B2);;break;
141:                                  case 2:
142:                                     output_low(A1);
143:                                     output_low(A2);
144:                                     output_high(B1);
145:                                     output_high(B2);;break;
146:                                  case 3:
147:                                     output_high(A1);
148:                                     output_low(A2);
149:                                     output_low(B1);
150:                                     output_high(B2);;break;
151:                           }
152:                           aux2 = numero_pasos;
153:
```

```
154:                        //Escritura en la EEPROM posición actual
155:                        guarda_eeprom(0x01, numero_pasos);
156:                        guarda_eeprom(0x02, (numero_pasos>>8));
157:
158:                        imprimir_posicion_actual();
159:
160:
                            if(( (!(input(FIND)))&& (dir=='R') ) || ((!(input(FIN
I)))&& (dir=='L'))){
161:
                               goto FIN;//SALTO  QUE SE EJECUTA SI SE CUMPLE ALG
UNA DE LAS CONDICIONES ANTERIORES
162:                        }
163: }
164: FIN:
165: aux2 = numero_pasos;
166:
167: }
168:
169: //SUBRUTINA QUE MUEVE EL DISPOSITIVO HASTA EL FINAL DE CARRERA DE LA IZQU
IERDA
170:
171: void posicion_izquierda(){
172:                    while(input(FINI)){
173:                        paso_motor(velocidad, 'L', 1);
174:                    }
175: }
176:
177: //SUBRUTINA QUE MUEVE EL DISPOSITIVO HASTA EL FINAL DE CARRERA DE LA DERE
CHA
178: void posicion_derecha(){
179:                    while(input(FIND)){
180:                        paso_motor(velocidad, 'R', 1);
181:                    }
182:                    extremo_derecho=aux2;
183:                    guarda_eeprom(0x03, extremo_derecho);
184:                    guarda_eeprom(0x04, (extremo_derecho>>8));
185: }
186:
187: //SUBRUTINA PARA PROBAR LA LCD
188:
189: void test_lcd(void){
190:                int i;
191:                lcd_putc("\f");
192:                lcd_gotoxy(1,1);
193:                lcd_command(BLINK_ON);
194:                lcd_command(BLINK_OFF);
195:
196:                for(i=0;i<sizeof(test1);i++){
197:                 lcd_gotoxy(i+1,1);
198:                 delay_ms(50);
199:                 printf(lcd_putc,"%c",test1[i]);
200: }
201:                delay_ms(100);
202:                for(i=0;i<sizeof(test2);i++){
203:                 lcd_gotoxy(i+1,2);
204:                 delay_ms(50);
205:                 printf(lcd_putc,"%c",test2[i]);
206:                }
207:                  delay_ms(100);
```

```
208:                    lcd_putc("\f");
209:                    lcd_command(BLINK_ON);
210: }
211:
212: //SUBRUTINA DE CONFIGURACION DEL MICROCONTROLADOR
213:
214: void configuracion (void){
215:
216:                    //PUERTOS DEL MICROCONTROLADOR
217:                    set_tris_a(0b01110111); //1 input; 0 output
218:                    set_tris_b(0b00000000);
219:                    set_tris_c(0b00000000);
220:
221:                    //TEMPORIZADORES Y CONTADORES
222:                    setup_counters(RTCC_INTERNAL,RTCC_DIV_256);
223:                    setup_timer_0(RTCC_INTERNAL);
224:                    setup_timer_1(T1_DISABLED);
225:                    setup_timer_2(T2_DISABLED,0,1);
226:                    setup_wdt(WDT_OFF);
227:
228:                    //PUERTOS ADC Y COMPARADORES
229:                    setup_adc_ports(AN0_TO_AN1_ANALOG|VSS_VDD);
230:                    setup_adc(ADC_CLOCK_DIV_64);
231:                    setup_comparator(NC_NC_NC_NC);
232:                    setup_vref(FALSE);
233:
234:                    //COMUNICACIONES
235:                    setup_spi(SPI_SS_DISABLED);
236:
237:                    //NTERRUPCIONES
238:                    disable_interrupts(int_ext);
239:                    disable_interrupts(INT_RTCC);
240:
                       disable_interrupts(GLOBAL);//DESHABILITO INTERRUPCIONES
GLOBALES
241:
242:
243: }
244:
245: //SUBRUTINA PARA LEER LOS PULSADORES(A TRAVES DEL ADC)
246:
247: int leer_pulsadores(void){
248:                    int var1;
249:
250:
                       set_adc_channel(0);//Asi   Captura el valor teniendo com
o referencia Vcc y Gnd o sea de 0 a 5v
251:                    delay_ms(10);
252:                    var1 = read_adc();
253:
254:                    //PULSADOR ACTIVADO
255:
256:                    if(var1>200){//IZQUIERDA
257:                       return 2;
258:                    }
259:                    else
260:                       if(var1>110){//DERECHA DERECHA
261:                          return 1;
262:                       }
263:                       else
```

```
264:                            if(var1>70){//IZQUIERDA  IZQUIERDA
265:                                return 4;
266:                            }
267:                            else
268:                                if(var1>55){//DERECHA
269:                                    return 3;
270:                                }
271:                                else
272:                                    if(var1>45){
273:                                        return 5;//INTRO
274:                                    }
275:
276: }
277:
278: //SUBRUTINA PARA ENVIAR EL EXTREMO DERECHO AL LABVIEW
279:
280: void envio_fin_carril_labview (){
281:
282:                 if(modo==1){
283:                     if(usb_enumerated()){
284:
285:
                                data_H    =    extremo_derecho>>8;//DESPLAZO  LA P
ARTE ALTA DE AUX2 HACIA LA PARTE BAJA Y LUEGO LA GUARDO EN DATA_H PORQUE AL IG
UALAR POR DEFECTO COJEMOS LA PARTE BAJA
286:
                                data_L    =    extremo_derecho;//EL  PROPIO MICRO
CREA UNA VARIABLE INTERMEDIA Y NO BORRA AUX2, DESPUES IGUALAMOS LA PATE BAJA D
E AUX2 A DATA_L
287:                                codigo=0b10110011;
288:
                                    printf(usb_cdc_putc, "%c%c%c%c%c",STX,codigo,da
ta_H,data_L,ETX);
289:                                delay_ms(50);
290:                        }
291:                    }
292: }
293:
294: //SUBRUTINA QUE REALIZA UNA MEDICION DEL ANCHO DEL CARRIL
295:
296: void inicializar_carril(void){
297:
298:
299:
                        posicion_izquierda(); //NOS DESPLAZAMOS HACIA LA IZQU
IERDA HASTA ALCANZAR EL FINAL DE CARRIL
300:                        guarda_eeprom(0x01,0x00);
301:                        guarda_eeprom(0x02,0x00);
302:
                            posicion_actual();//INICIALIZAMOS EL EXTREMO IZQUIER
DO A CERO
303:
                            posicion_derecha();//NOS DESPLAZAMOS HACIA LA DERECH
A HASTA ALCANZAR EL FINAL DE CARRIL
304:                        //CALCULAMOS LA MITAD DEL CARRIL
305:                        resul=(extremo_derecho-(extremo_derecho/2))+80;
306:                        resul1=resul;
307:                        resul=resul/250;
308:                        resul1=resul1%250;
309:                        paso_motor(velocidad, 'L', resul1);
```

```
310:
311:                         for(i=0;i<resul;i++){
312:                               paso_motor(velocidad, 'L', 250);
313:                            }
314:                       paso_motor(velocidad, 'R', 80);
315:                       posicion_actual();
316:                       lcd_putc("\f");
317:                       lcd_gotoxy(1,2);
318:                       printf(lcd_putc,"EXT. DCHO:%lu", extremo_derecho);
319:                       delay_ms(5000);
320:
                         envio_posicion_labview ();//ENVIAMOS AL LABVIEW LA POS
ICION ACTUAL
321:                       delay_ms(1500);
322:
                         envio_fin_carril_labview ();//ENVIAMOS AL LABVIEW EL E
XTREMO DERECHO
323: }
324:
325: //SUBRUTINA PARA SABER SI EL USB ESTÁ CONECTADO
326:
327: void usb_conectado(){
328:
329: set_adc_channel(1);
330: delay_ms(10);
331: var2 = read_adc();
332:
333: if (var2>240 && modo==0){//SE VERIFICA QUE HAY TENSION PROCEDENTE DEL U
SB
334:
335:                            modo=1;
336:                            lcd_putc("\f");
337:                            lcd_gotoxy(1,1);
338:                            printf(lcd_putc,"USB detectado");
339:                            lcd_gotoxy(1,2);
340:                            printf(lcd_putc,"Conecte al PC");
341:                            delay_ms(100);
342:                            usb_init();
343:
                               //ESPERAMOS HASTA QUE LA APLICACION DESARROLLADA
 SE CONECTA CON EL MICRO
344:                            while(!usb_cdc_connected()) {}
345:                            delay_ms(1500);
346:                            //ENVIAMOS LA POSICION ACTUAL AL LABVIEW
347:                            envio_posicion_labview();
348:                            lcd_putc("\f");
349:                            lcd_gotoxy(1,1);
350:                            printf(lcd_putc,"Pos.actual:%ld", aux2);
351:                            lcd_gotoxy(1,2);
352:                            printf(lcd_putc,"Pulse");
353:                            delay_ms(1500);
354:                            //ENVIAMOS EL FINAL DE CARRIL AL LABVIEW
355:                            envio_fin_carril_labview ();
356:   }
357:
358:
                          if(var2<240){//SE VERIFICA QUE NO HAY TENSIÓN PROCE
DENTE DEL USB
359:                            modo = 0;
360:
```

```
361:                       }
362: }
363:
364: //SUBRUTINA QUE DECODIFICA Y EJECUTA LAS ORDENES RECIBIDAS DEL LABVIEW
365:
366: void acciones_usb(){
367:
368: if  ((modo==1)){
369:
                     if(usb_cdc_kbhit() ){//SE VERIFICA SI HAY DATO EN EL B
UFFER
370:                       start_labview = usb_cdc_getc();
371:                       if(start_labview==STX){
372:
                                    //SI EL COMIENZO DE TRAMA COINCIDE CON
EL PROTOCOLO SE LEEN LOS 4 BYTES RESTANTES
373:                                  codigo_labview= usb_cdc_getc();
374:                                  data_H_labview= usb_cdc_getc();
375:                                  data_L_labview= usb_cdc_getc();
376:                                  end_labview = usb_cdc_getc();
377:                       }
378:
                          if((start_labview==STX) && (end_labview==ETX)){//S
E VERIFICA SI EL COMIENZO DE TRAMA Y EL FINAL DE TRAMA COINCIDEN CON EL PR
OTOCOLO
379:
380:                                 //INICIALIZACION
                                    1011 PARTE ALTA DE CODIGO
381:
                                     if( (bit_test(codigo_labview,7)) && (!bit_te
st(codigo_labview,6)) && (bit_test(codigo_labview,5)) && (bit_test(codigo_labvi
ew,4))){
382:                                 pulsador=6;
383:                                 }
384:
385:                                 //MOVERSE A LA IZQUIERDA 1010
386:
                                    //IGUALO LA POSICION A LA QUE IR Y LUEGO
EJECUTO UN INTRO
387:
                                    if( (bit_test(codigo_labview,7)) && (!bit_test(
codigo_labview,6)) && (bit_test(codigo_labview,5)) && (!bit_test(codigo_labview,4))
){
388:
                                      poslabview = data_H_labview;//GUARDAMOS  L
A VARIABLE DEL LABVIEW EN LA PARTE BAJA DE POSLABVIEW
389:
                                      poslabview=poslabview<<8;//DESPLAZO  8 BIT
S DE LA PARTE BAJA A LA PARTE ALTA   (PARTE ALTA|PARTE BAJA)
390:
                                      poslabview = poslabview | data_L_labview;
//SUMA  POR EL METODO DE OR
391:
392:                                   if(poslabview < aux2){
393:                                   pos=poslabview;
394:                                   pulsador=5;
395:                                   }
396:                                }
397:
398:                                //MOVERSE A LA DERECHA 0101
```

```
399:
                                    //IGUALO LA POSICION A LA QUE QUIERO IR Y
 LUEGO EJECUTO UN INTRO
400:
                                   if( !bit test(codigo_labview,7) && bit test
(codigo_labview,6) && !bit_test(codigo_labview,5) && bit_test(codigo_labview,4
)){
401:
                                     poslabview = data_H_labview;//GUARDAMOS  L
A VARIABLE DEL LABVIEW EN LA PARTE BAJA DE POSLABVIEW
402:
                                     poslabview=poslabview<<8;//DESPLAZO  8 BIT
S DE LA PARTE BAJA A LA PARTE ALTA   (PARTE ALTA|PARTE BAJA)
403:
                                     poslabview = poslabview | data_L_labview;
//SUMA  POR EL METODO DE OR
404:
405:                                   if(poslabview > aux2){
406:                                   pos=poslabview;
407:                                   pulsador=5;
408:                                   }
409:                                  }
410:
411:                                  // CODIFICACION DE VELOCIDAD 0       0000
412:
                                     if( !bit_test(codigo labview,3) && !bit test
(codigo_labview,2) &&  !bit_test(codigo_labview,1) &&  !bit_test(codigo_labview,
0)){
413:
414:                                  velocidad=500;
415:                                  }
416:                                  // CODIFICACION DE VELOCIDAD 1       0001
417:
                                     if( !bit_test(codigo_labview,3) && !bit_test
(codigo_labview,2) &&  !bit_test(codigo_labview,1) &&  bit_test(codigo_labview,0
)){
418:
419:                                  velocidad=400;
420:                                  }
421:                                  // CODIFICACION DE VELOCIDAD 2       0010
422:
                                     if( !bit test(codigo_labview,3) && !bit_test
(codigo_labview,2) &&  bit_test(codigo_labview,1) &&  !bit_test(codigo_labview,0
)){
423:
424:                                  velocidad=300;
425:                                  }
426:                                  // CODIFICACION DE VELOCIDAD 3       0011
427:
      if( !bit test(codigo_labview,3) && !bit test(codigo_labview,2) && bit_t
est(codigo_labview,1) && bit_test(codigo_labview,0)){
428:
429:                                  velocidad=250;
430:                                  }
431:                                  // CODIFICACION DE VELOCIDAD 4       0100
432:
                                     if( !bit test(codigo_labview,3) && bit_test
(codigo_labview,2) && !bit_test(codigo_labview,1) &&  !bit_test(codigo_labview,
0)){
433:
434:                                  velocidad=200;
```

```
435:                                }
436:                                // CODIFICACION DE VELOCIDAD 5      0101
437:
                                    if( !bit test(codigo_labview,3) && bit test
(codigo_labview,2) && !bit_test(codigo_labview,1) && bit_test(codigo_labview,0
)){
438:
439:                                velocidad=180;
440:                                }
441:                                // CODIFICACION DE VELOCIDAD 6      0110
442:
                                    if( !bit_test(codigo_labview,3) && bit test
(codigo_labview,2) && bit_test(codigo_labview,1) && !bit_test(codigo_labview,0
)){
443:
444:                                velocidad=160;
445:                                }
446:                                // CODIFICACION DE VELOCIDAD 7      0111
447:
                                    if( !bit_test(codigo_labview,3) && bit_test
(codigo_labview,2) && bit_test(codigo_labview,1) && bit_test(codigo_labview,0)
){
448:
449:                                velocidad=140;
450:                                }
451:                                // CODIFICACION DE VELOCIDAD 8      1000
452:
                                    if( bit test(codigo_labview,3) && !bit_test
(codigo_labview,2) && !bit_test(codigo_labview,1) && !bit_test(codigo_labview,
0)){
453:
454:                                velocidad=120;
455:                                }
456:                                // CODIFICACION DE VELOCIDAD 9      1001
457:
                                    if( bit test(codigo_labview,3) && !bit test
(codigo_labview,2) && !bit_test(codigo_labview,1) && bit_test(codigo_labview,0
)){
458:
459:                                velocidad=100;
460:                                }
461:                                // CODIFICACION DE VELOCIDAD 10      1010
462:
                                    if( bit_test(codigo_labview,3) && !bit test
(codigo_labview,2) && bit_test(codigo_labview,1) && !bit_test(codigo_labview,0
)){
463:
464:                                velocidad=80;
465:                                }
466:                                // CODIFICACION DE VELOCIDAD 11      1011
467:
                                    if( bit_test(codigo_labview,3) && !bit_test
(codigo_labview,2) && bit_test(codigo_labview,1) && bit_test(codigo_labview,0)
){
468:
469:                                velocidad=60;
470:                                }
471:                                // CODIFICACION DE VELOCIDAD 12      1100
472:
                                    if( bit_test(codigo_labview,3) && bit_test(
```

```
codigo_labview,2) && !bit_test(codigo_labview,1) && !bit_test(codigo_labview,0
)){
473:
474:                                velocidad=40;
475:                                }
476:                                // CODIFICACION DE VELOCIDAD 13      1101
477:
                                   if( bit_test(codigo_labview,3) && bit_test(
codigo_labview,2) && !bit_test(codigo_labview,1) && bit_test(codigo_labview,0)
){
478:
479:                               velocidad=20;
480:                               }
481:                               // CODIFICACION DE VELOCIDAD 14      1110
482:
                                   if( bit test(codigo labview,3) && bit_test(
codigo_labview,2) && bit_test(codigo_labview,1) && !bit_test(codigo_labview,0)
){
483:
484:                               velocidad=10;
485:                               }
486:                               // CODIFICACION DE VELOCIDAD 15      1111
487:
                                   if( bit test(codigo_labview,3) && bit test(
codigo_labview,2) && bit_test(codigo_labview,1) && bit_test(codigo_labview,0))
{
488:
489:                               velocidad=5;
490:                               }
491:                          }
492:                      }
493:                  }
494: }
495:
496: //SUBRUTINA QUE EJECUTA LAS ORDENES RECIBIDAS DE LOS PULSADORES Y DEL LAB
VIEW SOLO EN EL CASO 5 Y 6
497:
498: void acciones_manuales(){
499:
500:                     switch(pulsador){
501:
502:                         //MOVIMIENTOS A DERECHAS LARGOS
503:                         case 1:
504:                            if(pos>=(extremo_derecho-
5) && pos<=(extremo_derecho)){
505:                                pos = extremo_derecho;
506:                            }
507:                            else{
508:                               if(input(FIND))
509:                                  pos=pos+5;
510:                            }
511:                            delay_ms(10);
512:                            lcd_putc("\f");
513:                            lcd_gotoxy(1,1);
514:                            printf(lcd_putc,"Pos.actual:%lu", aux2);
515:                            lcd_gotoxy(1,2);
516:                            printf(lcd_putc,"Pos.nueva: %lu", pos);
517:                             delay_ms(10);
518:                         break;
519:
```

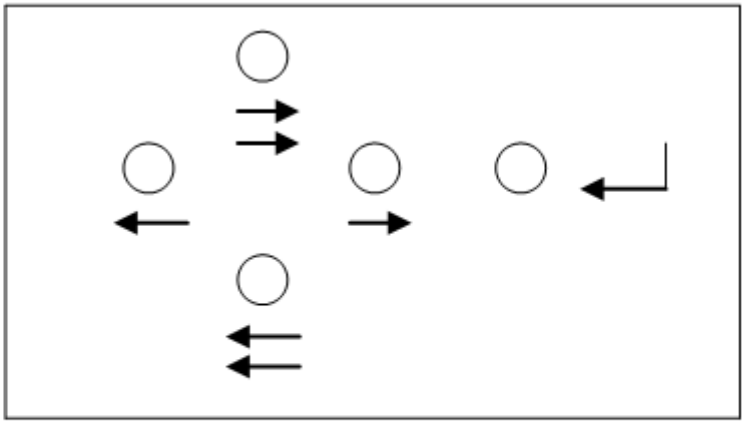
```
520:                            //MOVIMIENTOS A IZQUIERDAS CORTOS
521:                            case 2:
522:
                                   //SI NO SE HA LLEGADO AL EXTREMO IZQUIERDO S
E MUEVE LA VAGONETA A LA IZQUIERDA
523:                               if((input(FINI)) && (aux2>=0)){
524:                                  paso_motor(velocidad, 'L', 1);
525:                                  pos=aux2;
526:                                  delay_ms(100);
527:                               }
528:                               //SI SE HA LLEGADO AL EXTREMO IZQUIERDO
529:                               else{
530:                                  lcd_gotoxy(1,2);
531:                                  printf(lcd_putc,"Final carril izq");
532:                                }
533:                               envio_posicion_labview();
534:                               delay_ms(50);
535:                               break;
536:
537:                            //MOVIMIENTOS A DERECHAS CORTOS
538:                            case 3:
539:
                                   //MOVEMOS LA VAGONETA A LA DERECHA SI NO HEM
OS LLEGADO AL FINAL DE CARRIL DERECHO
540:
                                   if((input(FIND)) && (aux2 < extremo_derecho))
{
541:                                  paso_motor(velocidad, 'R', 1);
542:                                  pos=aux2;
543:                                  delay_ms(100);
544:                               }
545:
                                   //SI HEMOS LLEGADO AL FINAL DEL CARRIL DEREC
HO
546:                               else{
547:                                  lcd_gotoxy(1,2);
548:                                  printf(lcd_putc,"Final carril dch");
549:                               }
550:                               envio_posicion_labview();
551:
552:                               delay_ms(50);
553:
                                   //Indicar al PC que se ha movido un paso a
la derecha
554:                            break;
555:
556:                            //MOVIMIENTOS A IZQUIERDAS LARGOS
557:                            case 4:
558:                               if((pos>=0)&& (pos<=5) )
559:                                  pos = 0;
560:                               else{
561:                                  if(input(FINI)){
562:                                    pos=pos-5;}
563:                               }
564:                               delay_ms(10);
565:                               lcd_putc("\f");
566:                               lcd_gotoxy(1,1);
567:                               printf(lcd_putc,"Pos.actual:%lu", aux2);
568:                               lcd_gotoxy(1,2);
569:                                printf(lcd_putc,"Pos.nueva: %lu", pos);
```

```
570:                            delay_ms(10);
571:                         break;
572:
573:
                             //SE EJECUTAN LOS MOVIMIENTOS LARGOS O LOS RECIB
IDOS POR EL LABVIEW
574:                         case 5:
575:
576:
                                if(aux2<pos){//SI   LA POSICION ACTUAL ES MENO
R QUE LA FUTURA, SE REALIZA EL MOVIMIENTO A DERECHAS
577:
                                   if((input(FIND)) && (pos  <=extremo_derecho)){/
/SI NO HEMOS LLEGADO A TOCAR EL FINAL DE CARRIL Y LA POSICION FUTURA ES MENOR
QUE EL VALOR GUARDADO DEL EXTREMO DERECHO
578:
                                      //ENVIAMOS A LA VAGONETA A LA POSICION
 DESTINO
579:                                  resul=(pos-aux2);
580:                                  resul1=resul;
581:                                  resul=resul/250;
582:                                  resul1=resul1%250;
583:                                  paso_motor(velocidad, 'R', resul1);
584:
585:                                   for(i=0;i<resul;i++){
586:                                         paso_motor(velocidad, 'R', 250);
587:                                    }
588:                                     pos=aux2;
589:                                  }
590:                            }
591:
592:                            else{//SI  EL MOVIMIENTO ES A IZQUIERDAS
593:
                                   if(aux2>pos){//SI LA POSICION DESTINO ES
MENOR QUE LA ACTUAL
594:
                                      if(pos>80){//SI   LA POSICION FUTURA E
S MAYOR QUE 80 PASOS (REAJUSTE DEL DESTENSADO DE LA CORRE)
595:
                                        //SE CACULA LA POSICION FUTURA (HAC
IA LA IZQUIERDA) MENOS 80 PASOS Y SE ENVIA ALLI LA VAGONETA
596:                                    resul=(aux2-pos)+80;
597:                                    resul1=resul;
598:                                    resul=resul/250;
599:                                    resul1=resul1%250;
600:                                    paso_motor(velocidad, 'L', resul1);
601:                                    for(i=0;i<resul;i++){
602:                                       paso_motor(velocidad, 'L', 250);
603:                                     }
604:
                                     paso_motor(velocidad, 'R', 80);//NOS
  DESPLAZAMOS 80 PASOS A LA DERECHA PARA REALIZAR EL REAJUSTE
605:                                    pos=aux2;
606:
607:                                  }
608:
                                    else{//SI LA POSICION FUTURA ES MENOR
A 80, TOCAMOS EL FINAL DE CARRIL IZQUIERDO Y LUEGO VAMOS A LA POSICION DEST
INO
609:                                     posicion_izquierda();
```

```
610:                                          paso_motor(velocidad, 'R', pos);
611:                                            pos=aux2;
612:                                       }
613:                                     }
614:                                   }
615:                                 }
616:
617:                                   envio_posicion_labview();
618:
619:
620:                          break;
621:
622:
                              //SE EJECUTA UNA INICIALIZACION DEL CARRIL ORDEN
ADO POR EL LABVIEW
623:                          case 6:
624:                           inicializar_carril();
625:                           imprimir_posicion_actual();
626:
627:                         break;
628:                        }
629: }
630:
631:
632: //PROGRAMA PRINCIPAL MODULAR
633: void main(){
634:  configuracion();//INICIALIZAMOS EL MICROCONTROLADOR
635:  lcd_init();//INICIALIZAMOS LA LCD
636:  test_lcd();
637:  output_high(PIN_C0);
638:  modo=0;
639:
      //SE COMPRUEBA SI EL PULSADOR 5 ESTA PULSADO EN CUYO CASO SE EJECUTAR
Á UNA INICIALIZACION DEL CARRIL
640:  if(leer_pulsadores() == 5){
641:       posicion_actual();
642:       inicializar_carril();
643:       }
644:       else{
645:       posicion_actual();
646:       }
647:       imprimir_posicion_actual();
648:
649:
      while(TRUE){//BUCLE INFINITO QUE LLAMARÁ CICLICAMENTE A LAS FUNCIONES
POSTERIORES
650:
          pulsador = leer_pulsadores();//SE LEE EL ADC Y SE IGUALA A PULSAD
OR
651:      usb_conectado();//SE COMPRUEBA SI EL USB ESTÁ CONECTADO
652:      acciones_usb();//SE DECODIFICAN LAS ACCIONES DEL LABVIEW
653:
          acciones_manuales();//SE EJECUTA LAS ACCIONES DE LOS PULSADORES O
DEL LABVIEW
654:   }
655: }
```

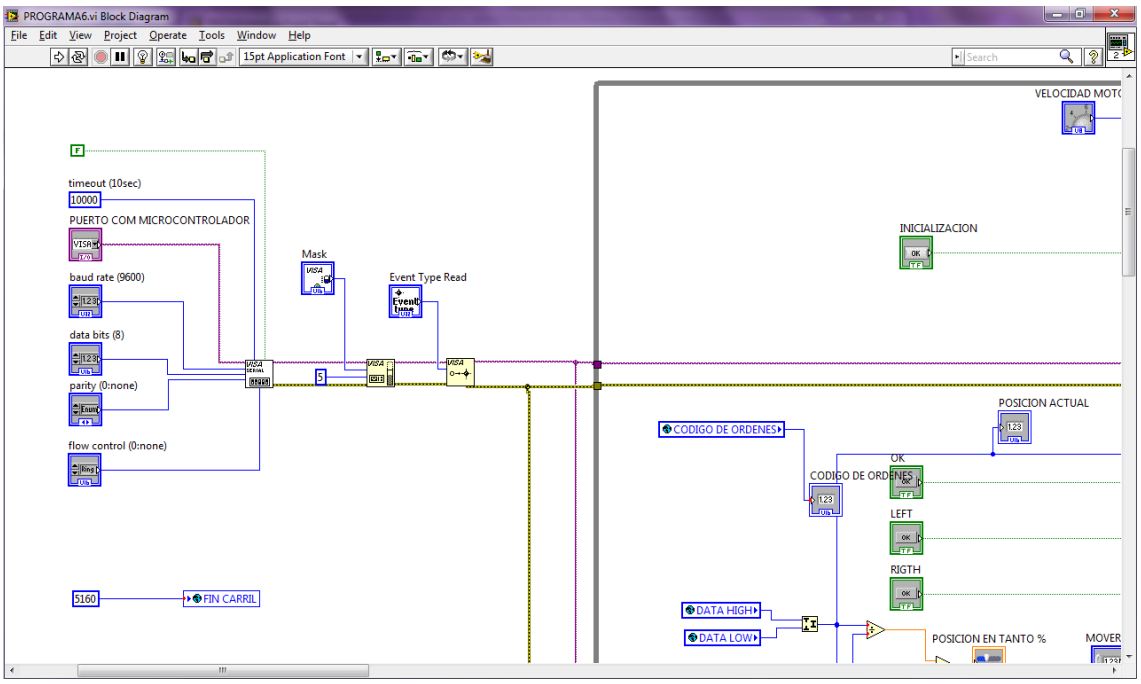
PLANOS

ANEXO 1: PEGATINAS INFORMATIVAS DE LOS PULSADORES

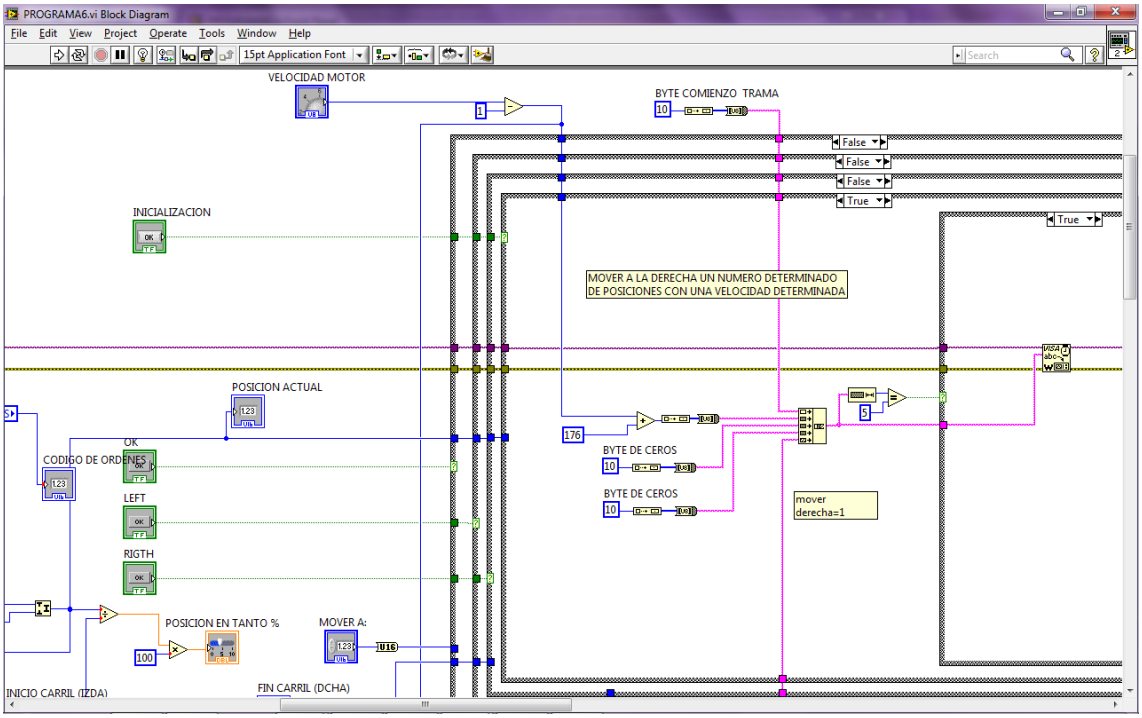


ANEXO2: CAPTURAS IMPORTANTES DEL *BLOCK DIAGRAM* DEL LABVIEW

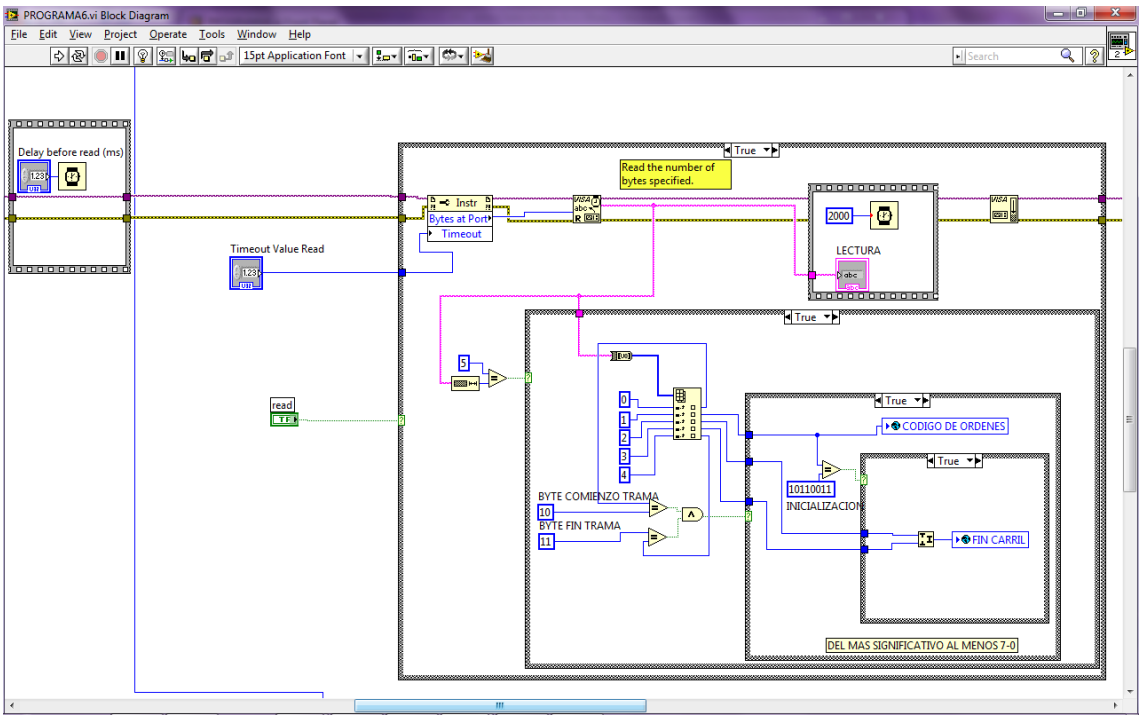
BLOQUE DE INICIALIZACION DE PUERTO SERIE



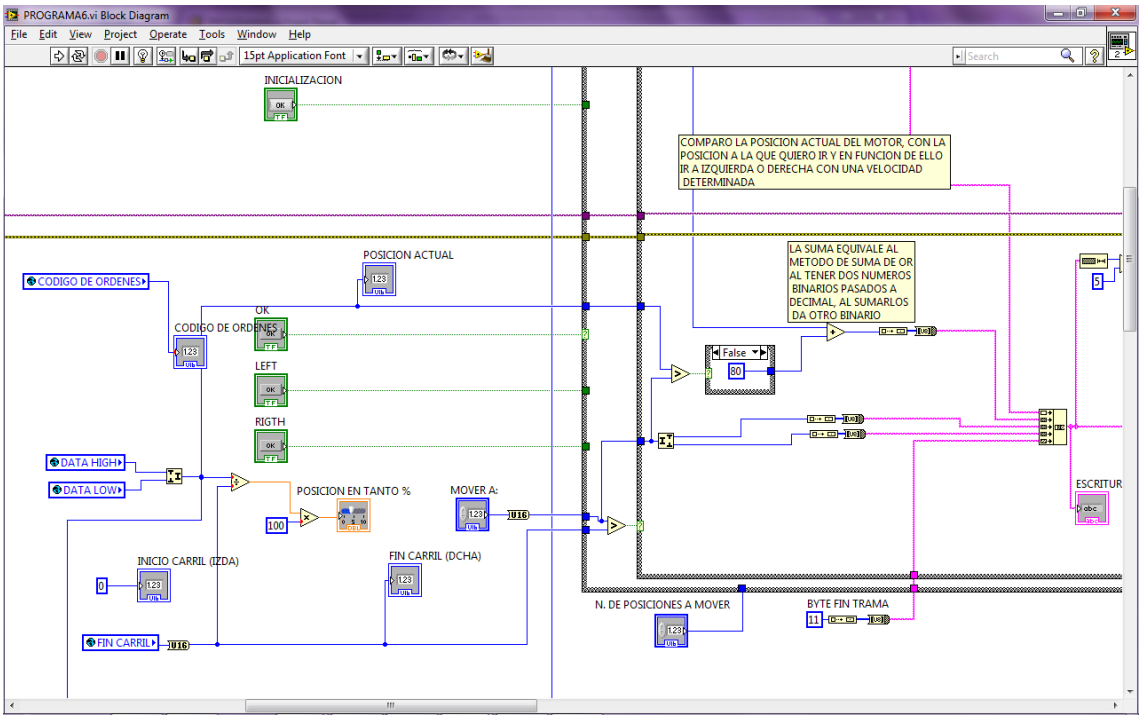
BLOQUE DE ESCRITURA EN EL PUERTO SERIE



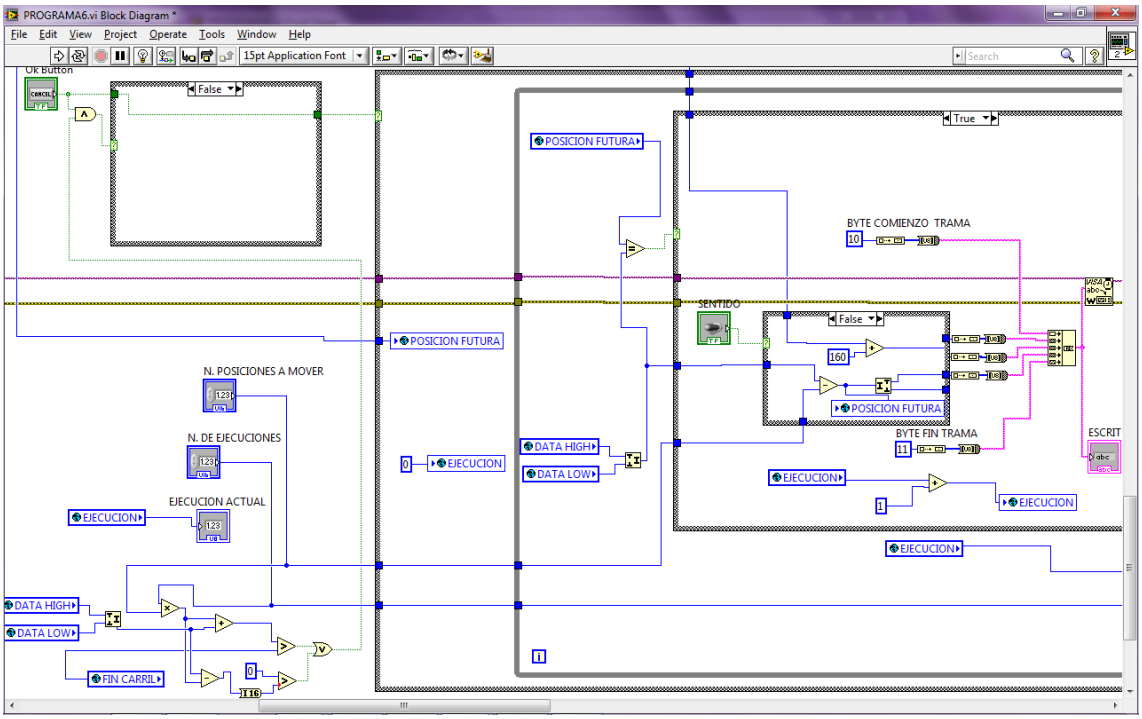
BLOQUE DE LECTURA DEL PUERTO SERIE



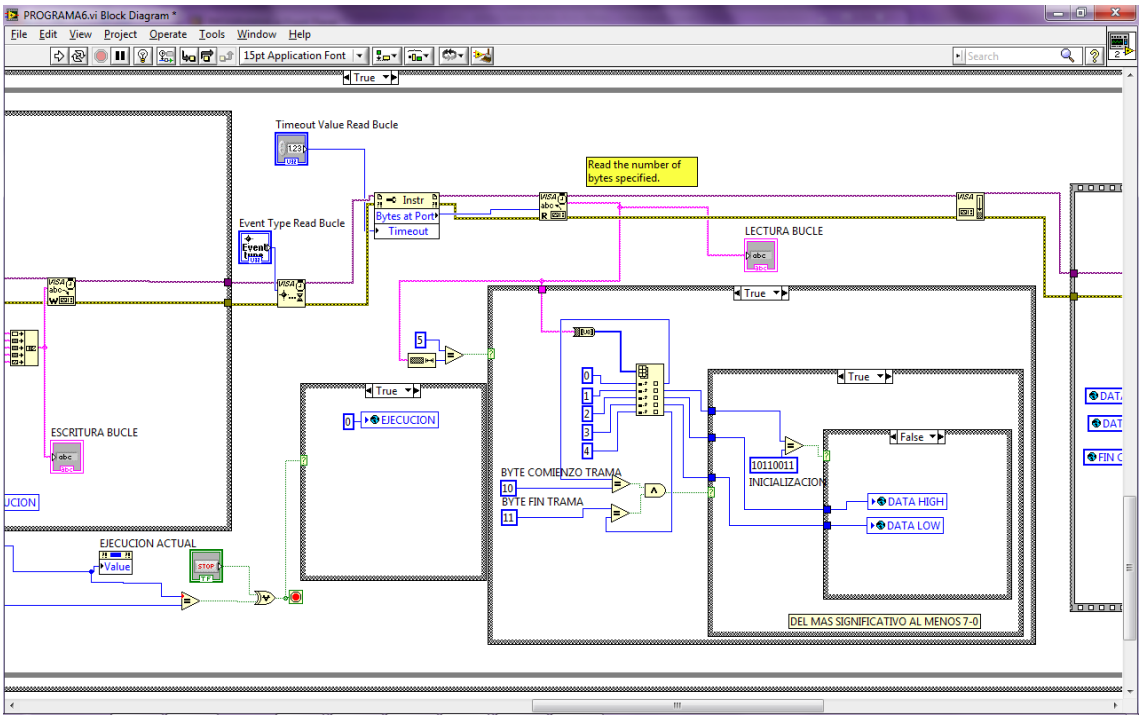
BLOQUE DE LECTURA DE VARIABLES GLOBALES



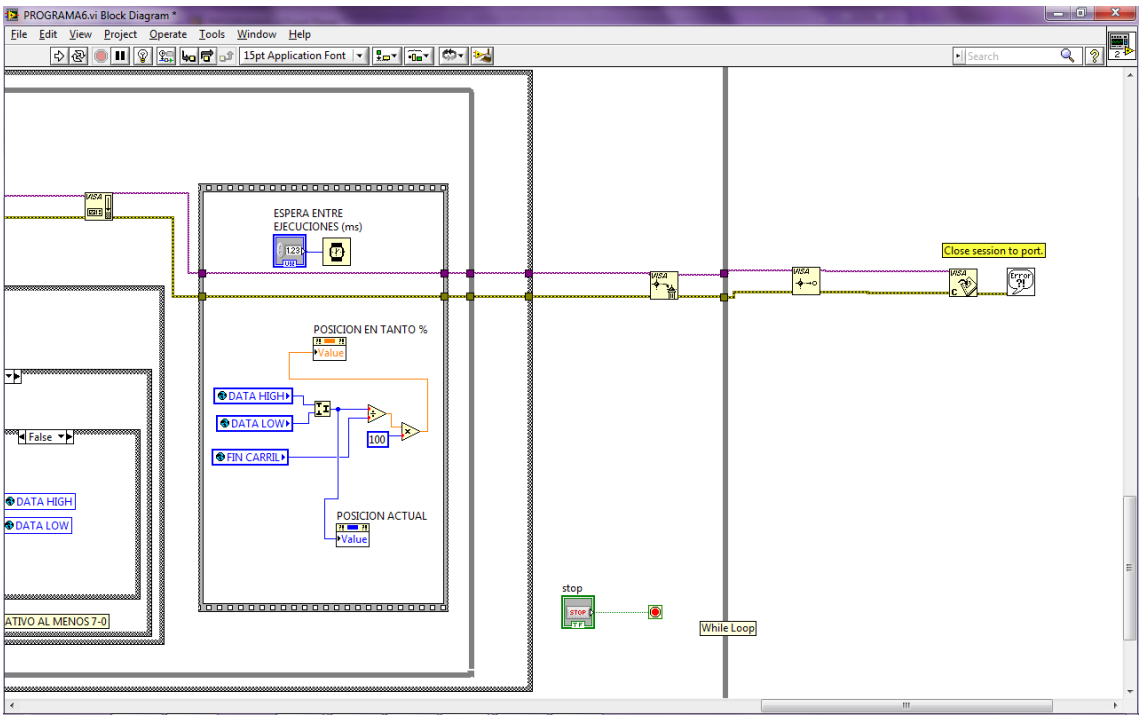
BLOQUE DE ESCRITURA DEL BUCLE



BLOQUE DE LECTURA POR INTERRUPCION DEL BUCLE



BLOQUE DE ESPERA DEL BUCLE Y CIERRE DEL PUERTO SERIE



PLIEGO DE CONDICIONES

PRESUPUESTO

1. INTRODUCCION

En este presupuesto se muestra una estimación de los gastos que ha conllevado la realización de este proyecto, quedando reflejados los costes del curso de LabVIEW que el ingeniero junior ha realizado, así como los relativos al personal que ha participado en el desarrollo.

2. COSTE DEL MATERIAL

Estos costes son los referentes al pago de la licencia del software LabVIEW y los materiales utilizados para realizar las mejoras mecánicas. La licencia ha sido comprada por el departamento de "Tecnología Electrónica" y ésta incluye instalaciones ilimitadas en las PCs del departamento. Por tanto, se ha estimado que se instalará el software en unos 10 PCs, hallándose el precio unitario por instalación.

Material mecánico:

Concepto	Unidades	Precio Unitario	Precio Total
Regletas	10	0,20 €	2,00 €
Laminas metalicas	2	5,00 €	10,00 €
Separadores	10	1,00 €	10,00 €
Tornillos	18	0,20 €	3,60 €
Tuercas	10	0,10 €	1,00 €
		TOTAL	**26,60 €**

Licencia de software:

Concepto	Número de licencias	Precio/Licencia	Subtotal
Licencia del software LabVIEW para el departamento de Tecnología Electrónica	1	7.699,00 €	7.699,00 €
		TOTAL	**7.699,00 €**

Concepto	Estimacion del numero de ordenadores en los que se instalará la licencia	Precio de la Licencia	Precio unitario
Estimacion del precio del software para un ordenador	10	7.699,00 €	769,90 €
		TOTAL	**769,90 €**

3. COSTE DEL PERSONAL INTERVINIENTE

Los costes que se van a detallar a continuación comprenden los costes desprendidos de las etapas de formación del ingeniero, diseño, desarrollo y pruebas (desarrolladas por un técnico) y los costes de redacción de la documentación relativa al proyecto.
Para la cuantificación del coste se ha considerado que el precio /hora del ingeniero junior es de 16€/hora, considerando que no dispone de experiencia en el desarrollo de proyectos.

El número de horas invertidas en el proyecto son:

Concepto	Numero de horas	Coste / hora	SubTotal
Curso de LabVIEW	120	10,00 €	1.200,00 €
Diseño del software de Control mediante LabVIEW	60	16,00 €	960,00 €
Diseño del nuevo software del microcontrolador	80	16,00 €	1.280,00 €
Depuracion y pruebas del sistema completo	40	16,00 €	640,00 €
		TOTAL	**4.080,00 €**

4. PRESUPUESTO TOTAL

El presupuesto de ejecución de este proyecto asciende a:

Concepto	Coste (€)
Costes de material	26,60 €
Costes del personal	4.080,00 €
Costes de la licencia	769,90 €
TOTAL	**4.876,50 €**

Asciende el presupuesto total de ejecución de este proyecto a la cantidad de CUATRO MIL OCHOCIENTOS SETENTA Y SEIS EUROS CON CINCUENTACENTIMOS. En esta cantidad está incluido el IVA exigido por Ley, con su valor actualizado.